WEITER-
BILDUNG

Programmierung des Mikroprozessorsystems U 880 D

Mikroprozessorsystem der II. Leistungsklasse

KDT-Lehrgang

Die technischen Angaben zu den Schaltkreisen des Mikroprozessorsystems U 880 D tragen reinen Informationscharakter.
Die ausgeführten Problembeispiele dienen der Demonstration der Einzelbefehle des Mikroprozessors und stellen damit keine Optimallösungen dar.

Anfragen und Hinweise, die den Inhalt dieses Lehrgangsmaterials betreffen, richten Sie bitte an

VEB Mikroelektronik Karl Marx Erfurt,
Beratungs- und Informationsstelle Mikroelektronik
5020 E r f u r t
Anger 22

Internes Lehrmaterial der Kammer der Technik.

Jede Vervielfältigung - auch auszugsweise - ist nur mit Genehmigung des Herausgebers gestattet.

Redaktionsschluß: 1. November 1983

V/4/56 R 122/84

KDT-Lehrgang

L e h r g a n g s m a t e r i a l

"Programmierung des Mikroprozessorsystems U 880 D"

Erarbeitet von einem
Autorenkollektiv der
Beratungs- und Informationsstelle
Mikroelektronik
des Bezirkes Erfurt

Trägerbetrieb:
VEB Mikroelektronik Karl Marx Erfurt
im VEB Kombinat Mikroelektronik

Vorwort

Die Durchsetzung des wissenschaftlich-technischen Fortschritts mit Hilfe der Mikroelektronik stellt eine der Quellen dar, mit deren Hilfe wir die dynamische Entwicklung unserer Volkswirtschaft sichern. Nur auf diese Weise schaffen wir die Grundlage für die weitere erfolgreiche Verwirklichung der Sozialpolitik und können den zusätzlichen Belastungen aus den veränderten äußeren Bedingungen wirksam begegnen. Das stellt hohe Ansprüche an alle, die wissenschaftlich-technische Aufgaben unter Anwendung der Mikroelektronik zu lösen haben.

Kein Zweifel besteht auch heute darüber, daß diese Forderung Betriebe aller Größenordnungen betrifft und nicht vor den Werktoren kleiner und mittelgroßer Betriebe Halt macht. Planmäßige und langfristige Lösungen - wie sie hier notwendig sind - erfordern entsprechend qualifizierte Kader.

Die Beratungs- und Informationsstelle Mikroelektronik im VEB Kombinat Mikroelektronik (BIS) sowie die Konsultationspunkte Mikroelektronik der Kammer der Technik haben in diesem Qualifizierungsprozeß wichtige Aufgaben zu erfüllen.

In Untersetzung der "Vereinbarung des 1. Sekretärs des Präsidiums der Kammer der Technik und des Generaldirektors des VEB Kombinat Mikroelektronik über eine Zusammenarbeit bei der Organisation der sozialistischen Gemeinschafts- und Bildungsarbeit zur Einsatzvorbereitung und umfassenden Anwendung der Mikroelektronik" wurden in sozialistischer Gemeinschaftsarbeit der BIS und der KDT-Bezirksvorstände der Bezirke Erfurt, Frankfurt/Oder und Leipzig Lernsysteme zur Programmierung des Mikroprozessorsystems U 880 D entwickelt und gebaut.

Mit dieser technischen Grundlage wird die Aus- und Weiterbildung praxisnah durchgeführt und auf ein höheres Niveau gehoben, als das bisher infolge fehlender materiell-technischer Voraussetzungen möglich war.

In diesen Lehrgängen (Dauer 8 Wochen/pro Woche 1 Tag) können jeweils 2 Lehrgangsteilnehmer gemeinsam ein Lernsystem zu ihrer Qualifizierung nutzen.

Ziel des Lehrganges ist es, die Teilnehmer innerhalb dieser, im Verhältnis zur Aufgabe kurzen Zeit an die Programmierung des Mikroprozessorsystems U 880 D heranzuführen und damit die Grundlage für eine erfolgreiche Analyse und Programmerstellung des im Anwenderbetrieb vorhandenen Einsatzproblems unter Nutzung der Mikroelektronik zu schaffen.

Das vorliegende Lehrgangsmaterial soll zur Unterstützung des Lernprozesses und zum Selbststudium dienen. Es erfüllt seinen Zweck, wenn es dazu beiträgt, den Lehrgangsteilnehmern bei der Aneignung der notwendigen Kenntnisse zu helfen und damit zur Erfüllung der in der Praxis gestellten Aufgaben Anleitung und Hilfe zu sein.

Bei der Überarbeitung des Lehrgangsmaterial wurden Hinweise von Lehrgangsteilnehmern aus den im Jahre 1983 durchgeführten Lehrgängen berücksichtigt.

KAMMER DER TECHNIK
Bezirksvorstand Erfurt

Beratungs- und
Informationsstelle
Mikroelektronik Erfurt

Prof.Dr.habil. Braune
Vorsitzender

Berthold
Bezirks-
sekretär

Dr.-Ing. Conrad
Leiter

Seite

Inhaltsverzeichnis

	Inhaltsverzeichnis	5
	Vorwort	3
1.	Grundbegriffe der Informationsdarstellung in Mikrorechnern	8
1.1.	Binärelemente	8
1.2.	Bit, Byte und Wort	8
1.3.	Digitale Codes	9
1.4.	Zahlensysteme	10
1.4.1.	Das reine Binärsystem	10
1.4.2.	Oktal- und Hexadezimalsystem	13
1.4.2.1.	Oktalsystem	14
1.4.2.2.	Das Hexadezimalsystem	14
1.5.	Darstellungsarten von Zahlen im Byte	15
1.5.1.	Darstellungsart positive Binärzahlen	15
1.5.2.	Darstellungsart Zweierkomplementzahl	16
1.5.3.	Darstellungsart gepacktes BCD-Format	17
1.6.	Übungen zur Wiederholung	17
2.	U 880 - Systembeschreibung	19
2.1.	Übersicht über das Mikroprozessorsystem U 880	19
2.2.	Aufbau und Arbeitsweise des Mikroprozessors U 880	24
2.2.1.	Registerstruktur	27
2.2.2.	Zeitverhalten der CPU U 880	32
2.2.3.	Interruptsystem	41
2.3.	Wiederholungsfragen	52
3.	Lernsystem	53
3.1.	Beschreibung der Konfiguration	53
3.1.1.	Allgemeines	53
3.1.2.	Zentraleinheit	53
3.1.3.	Speichereinheit	54

		Seite
3.1.4.	Eingabe- und Anzeigeeinheit	54
3.2.	Monitor-Kommandos	56
3.2.1.	Allgemeine Hinweise	56
3.2.2.	Funktionen der Datenbehandlung	56
3.2.3.	Funktionen für die Bedienungsorganisation, Daten- und Adresseneingabe und Register- und Speicheradressierung	57
3.2.4.	Funktionen für den Anwenderprogrammlauf- und -testung	59
3.2.5.	Funktionen für das Kassetten-Interface	59
3.3.	Hinweise zu einigen Funktionen	60
3.4.	Monitor-Bedienungsanweisung	61
3.5.	Universelle Unterprogramme	63
3.6.	Programmierhinweise	65
4.	Befehlsbeschreibung des U 880	66
4.1.	Befehlsstruktur	66
4.2.	Adressierungsarten	68
4.3.	Flag-Bit-Technik	70
4.4.	Befehlssatz des U 880	74
4.4.1.	8-Bit-Ladebefehle	74
4.4.2.	16-Bit-Ladebefehle	77
4.4.3.	Registertausch-Befehle	79
4.4.4.	8-Bit-Arithmetik-Befehle	82
4.4.5.	16-Bit-Arithmetik-Befehle	87
4.4.6.	Sprungbefehle	88
4.4.7.	Ein-Byte-Logik-Befehle	91
4.4.8.	Rotations- und Schiebebefehle	94
4.4.9.	Bitmanipulationsbefehle	100
4.4.10.	Blocktransfer- und Suchbefehle	100
4.4.11.	Spezielle Akkumulator- und Flagbefehle	105
4.4.12.	Indirekte Registeroperationen	105
4.4.13.	Unterprogrammaufruf- und Rücksprungbefehle	106
4.4.14.	Allgemeine Steuerbefehle	109

		Seite
5.	Programmierung der Peripheriebausteine	110
5.1.	Peripheriespezifische Softwaregrundlagen	110
5.1.1.	Ein-/-Ausgabebefehle	110
5.1.2.	Interrupts	114
5.2.	Parallel-Ein/Ausgabeschaltkreis U 855 D	119
5.2.1.	Kurzbeschreibung des U 855 D - PIO	119
5.2.2.	Betriebsarten des U 855 D - PIO	122
5.2.3.	Programmierung des U 855 D - PIO	123
5.3.	Zähler-/Zeitgabeschaltkreis U 857 D-CTC	127
5.3.1.	Kurzbeschreibung des U 857 D-CTC	127
5.3.2.	Betriebsarten des U 857 D-CTC	129
5.3.3.	Programmierung des U 857 D-CTC	129
6.	Programmbeispiele	135
7.	Das Mikrorechnersystem MRS K 1520	152
7.1.	Gerätetechnik des MRS K 1520	152
7.2.	K 1520 - Systemunterlagen	155
7.2.1.	Betriebssystem	155
7.2.1.1.	Betriebssystem für Mikrorechnerentwicklungssysteme	155
7.2.1.2.	BS für Datenverarbeitung	156
7.2.1.3.	BS zur Steuerung von Echtzeitprogrammen	156
7.2.2.	Basis-MOS	157
7.2.2.1.	Geräteroutinen	157
7.2.2.2.	Standardmodule	158
7.2.3.	Cross-Systemunterlagen	158
8.	Literaturverzeichnis	159

1. Grundbegriffe der Informationsdarstellung in Mikrorechnern

Für das Verständnis der Arbeit eines Mikrorechners und insbesondere seiner Programmierung sind einige Grundkenntnisse über die Informationsdarstellung in digitalen Systemen notwendig, die im folgenden kurz erläutert werden sollen.

1.1. Binärelemente

Informationen, wie Zahlen, Buchstaben, Operationen, Symbole werden in digitalen Systemen meist mit Binärelementen dargestellt. Binärlemente sind Elemente mit zwei gleichberechtigten Zuständen. Binärlemente, die sich zur Verschlüsselung (Codierung) von Informationen eignen, sind z. B. Schalter, die offen oder geschlossen sein können, Löcher oder keine Löcher in bestimmten Lochkarten- oder Lochbandstellungen, zwei mögliche Magnetisierungsrichtungen an einer Stelle auf einem Magnetband oder auf einer Magnetplatte, zwei verschiedene Frequenzen, zwei verschiedene Spannungs- oder Strompegel oder die abstrakten Symbole 0 oder 1 bzw. High oder Low.

1.2. Bit, Byte und Wort

Bit ist eine Abkürzung von "binary digit", das mit Binärziffer übersetzt werden kann. Das Bit ist die Grundeinheit der Information in der Digitaltechnik. Information wird meistens in der Form einer Bitfolge dargestellt. Die vierstellige Binärfolge 1000 kann z. B. die Dezimalzahl 8 darstellen. Der achtstelligen Binärfolge 11000001 entspricht der Buchstabe A im ASCII-Code.
Faßt man n-Bit zu einer Einheit zusammen, so spricht man von einem Wort mit einer Länge von n-Bit. Eine achtstellige Binärfolge oder auch Wort mit einer Länge von 8 Bit wird als Byte bezeichnet, während die 16-stellige Bitfolge, die

für die Adressierung in Mikrorechnern verwendet wird, häufig als Adreß-Wort bezeichnet wird.

1.3. Digitale Codes

Ein digitaler Code läßt sich als ein System von Symbolen verstehen, das es erlaubt, Daten so darzustellen, daß diese von einem Computer oder einem digitalen System verstanden, gespeichert und verarbeitet werden können. Es lassen sich verschiedene Codeklassen unterscheiden, die wichtigsten seien hier genannt:

- Codes, die in Digitalschaltungen verwendet werden, um bestimmte Operationen, z. B. Zählen, auszuführen. Ein Beispiel hierfür ist der Binärcode.

- Codes, mit denen die Dezimalziffern 0 9 binär verschlüsselt werden.
  Beispiele: Binär verschlüsselter Dezimalcode (BCD = Binary Codet Dezimal), binary Code

- Codes, die geeignet sind, Dezimalziffern, Buchstaben, Operationen, Steuerzeichen usw. in Binärform darzustellen.
  Beispiele: ASCII-Code, ISO-7 Bit - Code

- Codes, die dem Befehlssatz eines Computers entsprechen.
  Beispiele: U808-Befehlscodes, U880-Befehlscodes

Bei der Programmierung des U880-Mikroprozessors werden hauptsächlich der Binärcode, der BCD-Code, der ASCII-Code (American Standard Code for Information Interchange) und der U880-Befehlscode verwendet.

1.4. Zahlensysteme

1.4.1. Das reine Binärsystem

Wir sind gewöhnt, im Dezimalsystem zu denken. Die einzelnen Ziffern einer Zahl haben positionsabhängige Wertigkeiten, z. B.

$$1982 = 1 \cdot 10^3 + 9 \cdot 10^2 + 8 \cdot 10^1 + 2 \cdot 10^0$$
$$= 1 \cdot 1000 + 9 \cdot 100 + 8 \cdot 10 + 2 \cdot 1$$

Das Dezimalsystem umfaßt die Ziffern 0 ... 9, die Basis des Zahlensystems ist 10.
Das reine Binärsystem oder kurz Dualsystem umfaßt die Ziffern 0 und 1, die Basis ist 2. Die einzelnen Ziffern einer Zahl haben ebenfalls positionsabhängige Wertigkeiten. Zum Beispiel:

$$10111 = 1 \times 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$
$$= 1 \times 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1$$
$$= 23$$

Die Binärzahl 10111 entspricht der Dezimalzahl 23. Das kann auch kurz folgendermaßen geschrieben werden:

$$10111_2 = 23_{10}$$

Tabelle 1.1 Dezimal- und Binärzahlen 0 ... 16

Dezimalzahl	Binärzahl
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

8	1000
9	1001
10	1011
11	1010
12	1100
13	1101
14	1110
15	1111
16	1 0000

Die Tabelle 1.1. zeigt die Zuordnung zwischen einfachen Dezimal- und Binärzahlen. Mit 4 Binärziffern lassen sich also die 16 Dezimalzahlen 0 ... 15 darstellen. Dezimalzahlen, die größer sind, benötigen in der Binärdarstellung mehr als 4 Bit.

Tabelle 1.2. Ausgewählte Dezimal- und Binärzahlen zwischen 0 und 65535

Dezimalzahl	Binärzahl	
0	0	
1	1	
2	10	
3	11	
4	100	
7	111	
8	1000	
15	1111	4 Bit
16	10000	
31	11111	
32	100000	
63	111111	
64	1000000	
127	1111111	
128	10000000	
255	11111111	8 Bit
256	100000000	
511	111111111	
512	1000000000	
1023	1111111111	10 Bit
1024	10000000000	
2047	11111111111	
2048	100000000000	
4095	111111111111	
4096	1000000000000	
8191	1111111111111	
8192	10000000000000	

Dezimalzahl	Binärzahl	
16383	11111111111111	
16384	100000000000000	
32767	111111111111111	
32768	1000000000000000	
65535	1111111111111111	16 Bit

Einer 8-Bit-Binärzahl entsprechen die Dezimalzahlen 0...255. Damit lassen sich 256 verschiedene Zuordnungen, wie Instruktionen, Geräte, Impulse o. ä. unterscheiden. Der U880 Mikroprozessor hat ein 8-Bit-Datenbus und einen 16-Bit-Adressenbus. Mit der 16-Bit-Adresse können 2^{16} = 65535 Speicherplätze adressiert werden. Mit dem 8-Bit-Datenwort kann man 2^8 = 256 verschiedene I/O-Kanäle adressieren.

Zur Abkürzung wurde definiert:

$$1\ K = 1024 = 2^{10}$$

Somit lassen sich die adressierbaren Speicherplätze auch wie folgt ausdrücken:

2^{16} $2^{(6 + 10)} = 2^6$ $2^{10} = 2^6$ 1K = 64 1K

Da die Speicherwortbreite 8 Bit 1 Byte ist, spricht man auch von 64 K-Byte adressierbarem Speicherbereich.

1.4.2. Oktal- und Hexadezimalsystem

Weil die Handhabung der reinen Binärzahlen nicht sehr übersichtlich sind, werden speziell in der Mikrocomputertechnik das Oktalsystem und das Hexadezimalsystem verwendet, die eine wesentlich größere Übersicht bieten.

1.4.2.1. Das Oktalsystem

Das Oktalsystem verwendet die Ziffern 0 7, die Basis des Zahlungssystems ist $2^3 = 8$. Die Umwandlung von Binärzahlen in Oktanzahlen erfolgt so, daß man die Binärzahl von rechts nach links fortlaufend in Dreiergruppen aufteilt und jeder Dreiergruppe eine Oktalziffer entsprechend ihrem Gruppenwert zuordnet, z. B.

$$10011101 = 010\ 011\ 101 = 2\ 3\ 5_8$$

$$11000111 = 011\ 000\ 111 = 3\ 0\ 7_8$$

Eine 8-Bit-Binärzahl kann im Oktalsystem an der ersten Stelle höchstens eine 3 haben, da das fehlende 9. Bit durch eine Null ergänzt wird.

1.4.2.2. Das Hexadezimalsystem

Das Hexadezimalsystem wird teilweis e auch als Sedezimalsystem bezeichnet. Die Basis dieses Systems ist $16 = 2^4$. Das Hexadezimalsystem verwendet die Ziffern 0 9 und die Buchstaben A F.

Tabelle 1.3. Hexadezimal- und 4- und 8-Bit-Binärzahlen

Hexadezimalzahl	Binärzahl	Dezimalzahl
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9

Hexadezimalzahl	Binärzahl	Dezimalzahl
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15
10	00010000	16
1F	00011111	31
20	00100000	32
2F	00101111	47
F0	11110000	240
FF	11111111	255

Die Umwandlung von Binärzahlen in Hexadezimalzahlen erfolgt so, wie beim Oktalsystem, nur werden hier jeweils fortlaufend Vierergruppen eingeteilt. Zur Kennzeichnung einer Hexadezimalzahl wird der Zahl entweder die Fußnote 16 beigefügt oder der Buchstabe H angehängt.

1. 1001 1101 = 1001 1101 = $9D_{16}$ (oder 9DH)
2. 0011 1111 = 0011 1111 = $3F_{16}$ (oder 3FH)
3. 1011 0110 1011 = 1001 0110 1011 = $96B_{16}$ (oder 96BH)

1.5. Darstellungsarten von Zahlen im Byte

Es gibt verschiedene Arten, den binären Inhalt eines Bytes zu interpretieren, wobei jede Art für sich inbezug auf bestimmte Anwendungen vorteilhaft ist.

1.5.1. Darstellungsart positive Binärzahl

Interpretiert man den Inhalt eines Bytes als reine Binärzahl, so können Zahlen zwischen 0 und 255 dargestellt werden. Reine Binärzahlen werden normalerweise als positive Zahlen gewertet. Ein eventuelles Vorzeichen dazu wird speziell zugewiesen und separat gespeichert.

Die Anwendungsbreite dieser Darstellungsart reicht von der Darstellung von Zählereignissen über Codierungsarten von Informations- und Steuerzeichen bis zur Darstellung der Befehlscode des Mikroprozessors.

1.5.2. Darstellungsart Zweierkomplementzahl

Das Zweierkomplement ist für die Darstellung negativer Zahlen in einem Byte von Bedeutung. Dazu wird der Zahlenbereich der reinen Binärzahl, der von 0 bis 255 reicht, in zwei Teilbereiche aufgespalten, die unsymmetrisch zu 0 sind. Die beiden Teilbereiche umfassen dann - 128 bis 1 und 0 bis + 127. Das Zweierkomplement $\overline{Z}$ einer 8-stelligen Binärzahl Z ist definiert als die Ergänzung von Z zur Zweierpotenz 2^8

$$\overline{Z} = 2^8 - Z$$

Ist Z positiv, so gibt $\overline{Z}$ die Darstellung von - Z an und umgekehrt liefert $\overline{Z}$ bei negativem Z den zugehörigen positiven Wert.
Ein einfaches Verfahren zur Ermittlung des Zweierkomplementes besteht darin, die Binärzahl Z bitweise zu invertieren und anschließend 1 zu addieren.

Beispiel: 00011100 = 28_{10}

inv. 11100011

1

11100100 = $- 28_{10}$

Beispiel: 10011111 = $- 97_{10}$

inv. 01100000

1

01100001 = 97_{10}

Das höchstwertigste Bit eines Bytes in der Zweierkomplementdarstellung wird als Vorzeichen verwendet. 0 steht für positive und 1 für negative Zahlen.

1.5.3. Darstellungsart gepacktes-BCD-Format

Teilt man die 8 Bit eines Bytes in zwei Gruppen zu jeweils 4 Bit auf, ähnlich dem Einteilungsprinzip bei der Hexadezimalschreibweise, so können mit jedem Halbbyte der Wertebereich 0 ... 15 erfaßt werden. Wird der Wertebereich auf 0 ... 9 eingeschränkt, so spricht man von gepacktem-BCD-Format, weil jeweils 4 Bit eine Dezimalziffer zwischen 0 und 9 darstellen. In einem Byte können damit Zahlen zwischen 0_{10} und 99_{10} erfaßt werden. Bei der Verarbeitung von Bytes, die Zahlen im gepacktem-BCD-Format darstellen, sind die normalen Bytearithmetikbefehle des Mikroprozessors nur unter Anwendung eines zusätzlichen Dezimalkorrekturbefehls (DAA) arithmetisch richtig verwendbar.

Beispiele für gepacktes BCD-Format:

1. 0000 1000 = 8_{10}
2. 0110 0111 = 67_{10}
3. 1001 1001 = 99_{10}

1.6. Übungen zur Wiederholung

1. Welchen Dezimalzahlen entsprechen die folgenden Binärzahlen?

   a) 11101
   b) 11111111
   c) 1111111111111111
   d) 1001
   e) 11010011
   f) 10011

2. Welchen Oktalzahlen entsprechen die folgenden Binärzahlen?

a) 11010011
b) 00111110
c) 01110110
d) 00111100
e) 11111111
f) 00110010
g) 11000011
h) 00000010

3. Welchen Hexadezimalzahlen entsprechen den in Übung 2 angegebenen Binärzahlen?

4. Geben Sie für die in Übung 2 angegebenen Binärzahlen die dazugehörigen Zweierkomplementzahlen an.

2. U 880 - Systembeschreibung

2.1. Übersicht über das Mikroprozessorsystem

Das Mikroprozessorsystem U 880 besteht im wesentlichen aus folgenden Grundbausteinen:

U 880 D - CPU (Zentrale Verarbeitungseinheit)
U 855 D - PIO (Parallele Ein/Ausgabe)
U 856 D - SIO (Serielle Ein/Ausgabe)
U 857 D - CTC (Zähler/Zeitgeber)

Alle Schaltkreise werden in n-Kanal-Silicon-Gate-Technologie hergestellt und haben ein 40-poliges DIL-Plastgehäuse (Ausnahme CTC mit 28-poligem Gehäuse). Die Architektur der Schaltkreise realisiert ein einheitliches Systemkonzept, das folgende wesentliche Merkmale aufweist:

- Signalzuführung über zwei Systembusse (Adressenbus, Datenbus) und eine Reihe von Steuerleistungen
- einheitliches Interruptregime mit der Möglichkeit der Prioritätsverkettung der Peripherieschaltkreise

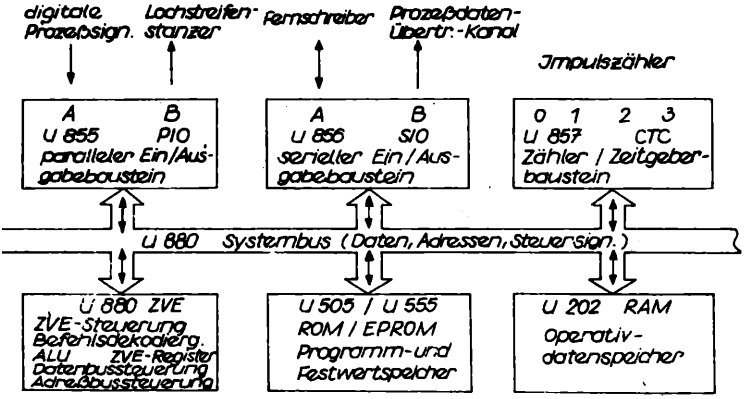


Bild 2.1 Grundstruktur eines mit Bausteinen aus dem System U 880 aufgebauten Mikrorechners

- hochgradige Softwareprogrammierbarkeit der schaltkreisspezifischen Eigenschaften
- gemeinsamer Systemtakt als Einphasentakt mit einer maximalen Frequenz von 2,5 MHz
- einheitliche Spannungsversorgung von + 5 V

Die U 880-ZVE übernimmt als Kernstück des gesamten Bausteinsortiments die Systemüberwachung, die Befehlsabarbeitung und den überwiegenden Teil der Interruptsteuerung. Es handelt sich hier um einen Digitalrechnerkern, der mit Befehlswörtern variabler Länge - 1, 2, 3 oder 4 Byte - standardmäßig Datenwörter von 8 bit, entsprechend einem Byte, aber auch von 1, 4 und 16 bit bearbeiten kann. Neben den Datenmanipulationsbefehlen stehen eine Fülle von Befehlen zur Programmorganisation zur Verfügung. Solche Operationen sind im allgemeinen Erzeugung, Transport, Kombination oder Manipulation von Datenwerten. Datenquellen oder Datensenken können dabei die CPU, die Peripherieschaltkreise oder die Speicher sein. Der Datentransfer innerhalb des Systems erfolgt über die CPU (Ausnahme: Datentransfer mittels DMA-Betrieb).

Die Befehlszykluszeit liegt zwischen 1,2 und 1,6 µs. Die einzelnen Instruktionen werden unterschiedlich, in einem oder in mehreren Befehlszyklen abgearbeitet. Die Möglichkeit, Einzel-Bit-Operationen (Steuerungstechnik/Programmorganisation), BCD-Operationen (Datentechnik) und 16-bit-Operationen (Prozeßrechentechnik) ausführen zu können, ist eine der wesentlichen Erweiterungen gegenüber dem Vorgänger U 808.

Bei dem U 855 handelt es sich um einen Interfacebaustein, der über zwei 8-bit-Datenkanäle (Ports) den parallelen Datenaustausch zwischen Mikroprozessor und Umwelt realisiert. Dieser parallele Ein-/Ausgabebaustein kann über

Steuerbefehle, die vom Mikroprozessor an den PIO ausgegeben werden, an die jeweilige Aufgabe (Betriebsart) angepaßt werden. Der U 855 kann sowohl beim Anschluß von Datenverarbeitungsperipherie - z. B. Lochstreifenleser/-Stanzer, Drucker- als auch zur Ankopplung von Prozeßsignalen - z. B. digitale Eingabe, digitale Ausgabe - Verwendung finden. Er enthält wie auch die anderen E/A-Bausteine eine Quittungsbetriebslogik (HANDSHAKE) und den restlichen nicht im Mikroprozessor befindlichen Teil des Interruptsystems. Der U 855-Ein-/Ausgabe-Baustein wird zusammen mit weiteren Peripheriebausteinen in eine Prioritätskaskade (DAISY CHAIN) eingebunden, die der im jeweiligen Einsatzfall gewünschten Vorrangstruktur der Peripheriegeräte entsprechen muß.

Der U 856 - SIO dient zur Ankopplung von Ein-/Ausgabe-Geräten mit bit-seriellem Interface an den Mikroprozessor. Solche Geräte sind u. a. Fernschreiber und Floppy-Disks, aber auch serielle Prozeßdatenübermittlungskanäle.
Der SIO-Baustein enthält zwei serielle Kanäle, die unabhängig voneinander sowohl Daten aus- (Sender) als auch eingeben (Empfänger) können. Die Datenübertragung zwischen Peripheriegerät und Interfacebaustein kann synchron oder asynchron erfolgen. Die Kommunikation zwischen dem U 856 und dem Mikroprozessor U 880 erfolgt wie beim U 855 - PIO in byteserieller (8bit-paralleler) Form. Die Umwandlung bit-serieller Datenwörter in byteserielle Datenwörter und umgekehrt, einschließlich einer doppelten 8-bit-Empfangspufferung, erfolgt in dem seriellen Ein-/Ausgabe-Baustein. Die maximale Datenübertragungsrate, eine wichtige Kenngröße jedes seriellen Datenkanals, liegt bei etwa 500 k bit/s. Das beim parallelen Ein-/Ausgabe-Baustein zur Quittungsbetriebslogik und zum Interruptsystem Gesagte gilt auch für den U 856.

Der U 857 - CTC ist ein als Zeitgeber oder als Zähler zu verwendender Peripherieschaltkreis. Er ermöglicht dem Anwender wahlweise die über ein Interruptsignal ausgelöste

Bearbeitung mehrerer (max. 4) zeitzyklischer Echtzeitaufgaben oder das Abarbeiten voreingestellter Rückwärtszähler mit Interruptmeldung bei Erreichen des Zählerstandes Null. Der CTC-Schaltkreis ist intern aus vier voneinander unabhängigen 8-bit-Kanälen aufgebaut. Die Zeitgeberfunktion des CTC-Schaltkreises ist prädestiniert für den Aufbau einer softwaregesteuerten Echzeituhr im Mikroprozessor. In der Betriebsart Zähler führen bis zu 256 externe Einzelimpulse jeweils zu einer Interruptmeldung im ZVE-Schaltkreis. Der U 857 - CTC-Schaltkreis kann auch im Abfrage-Verfahren (Polling) betrieben werden, wenn das Interruptsystem gesperrt werden muß. Außerdem ist es möglich, die vier unabhängigen CTC-Kanäle durch entsprechende Schaltungsmaßnahmen verkettet zu betreiben.

Für den Aufbau von Mikrorechnersystemen sind neben den Grundbausteinen auch Speicherbauelemente notwendig. Meist werden die generell byteorientierten Programmspeicher, die vor allem als Festwertspeicher (ROM, read only memory) ausgeführt sind, und die Datenspeicher, die als RAM-Lese-Schreib-Speicher ausgeführt sind, zu einem einheitlichen Arbeitsspeicher zusammengefaßt, wobei der Adreßraum für ROM- und für RAM-Bereiche genutzt wird.

Die Gruppe der Festwertspeicher ROM hat die Aufgabe, als Nur-Lese-Speicher einmal eingeschriebene Daten oder Befehle zerstörungsfrei für ein beliebig häufiges Lesen bereitzustellen.
Die wichtigsten Arten der Festwertspeicher sind:

- ROM Der eigentliche ROM ist ein im letzten Fertigungsschritt maskenprogrammierter Festwertspeicher mit nicht mehr veränderlichem Inhalt (Bitmuster).

- PROM (programable read only memory):
  Vom Anwender mit einem speziellen Programmiergerät programmierbare Festwertspeicher, deren eingeschriebener Inhalt ebenfalls nicht mehr gelöscht werden kann.

- EPROM (erasable PROM):
  Vom Anwender mit einem speziellen Programmiergerät elektrisch programmierbare Festwertspeicher, deren Inhalt mit Hilfe von UV-Licht global gelöscht werden kann (ca. 1001x).

Lese-Schreibspeicher (RAM, random access memory), Speicher mit wahlfreiem Zugriff, haben die Aufgabe, Daten oder Befehle während des Rechenbetriebes des Mikroprozessors aufzunehmen und wieder bereitzustellen.
Mit dem Abschalten der Betriebsspannung verlieren sie ihre Information, wenn nicht spezielle Maßnahmen zur Betriebsspannungspufferung vorgesehen sind. Bezüglich ihrer Systemeigenschaften unterscheiden sich RAM's nicht wesentlich, unabhängig in welcher Technologie sie hergestellt sind. Während ROM's fast ausschließlich Byte-organisiert sind, haben RAM's sowohl 1-Bit- als auch 4- und 8-Bit-Verarbeitungsbreite.

2.2. Aufbau und Arbeitsweise des Mikroprozessors U 880

Die Zentraleinheit des Mikroprozessorsystems U 880 ist neben dem leistungsfähigen Befehlsrepertoire und der hohen Verarbeitungsgeschwindigkeit durch einen umfangreichen Registersatz (Bild 2.2) gekennzeichnet.
Die CPU enthält 208 Bits im internen RAM, zu denen der Programmierer Zugriff hat. Bild 2.3 illustriert, wie dieser Speicher in 18 Register zu 8 Bit und 4 Register zu 16 Bit unterteilt ist.

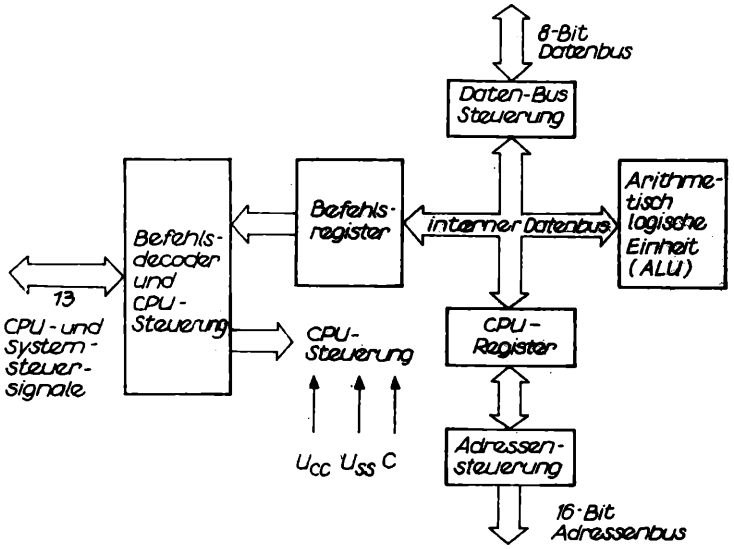


Bild 2.2 Blockschaltbild der CPU - U 880 D

Alle U 880-Register sind als statische RAMs ausgeführt. Die Register umfassen zwei Sätze von 6 Allgebrauchsregistern, die individuell als 8-Bitregister oder in Paaren als 16-Bitregister verwendet werden können. Ebenfalls sind zwei Akkumulatoren und zwei Flagregister vorhanden.

Hauptregistersatz		Alternativsatz		
Akkumulator A	Flags F	Akkumulator A'	Flags F'	
B	C	B'	C'	Register zur allgemeinen Verwendung
D	E	D'	E'	
H	L	H'	L'	

Interrupt Vektor I	Speicher Refresh R	
Index Register	IX	Spezialregister
Index Register	IY	
Keller-Zeiger	SP	
Programmzähler	PC	

Bild 2.3 Aufbau des CPU-Registers

Neben den allgemeinen Registern existieren zwei Indexregister (IX und IY), ein Stack-Pointer (SP) zur Lokalisierung des Stack im RAM-Bereich, ein Befehlszähler (PC), ein Interruptvektorregister (I) und ein Speicher-Refresh-Register (R). Die Kommunikation des Mikroprozessors mit seiner Umwelt läuft über einen 8-bit-Datenbus, über einen 16-bit-Adressenbus und über eine Reihe von Steuersignalen (Tafel 2.1).

Tafel 2.1 Signalbelegung U 880

Signal-bezeichnung	Erläuterung
$A_{15} \dots A_0$	Adressensignale 16 bit
$D_7 \dots D_0$	Datensignale 8 bit
MREQ	MEMORY REQUEST kennzeichnet, daß auf dem Adreßbus eine Speicher-Lese- oder eine Speicher-Schreib-Adresse ansteht.
IORQ	IN/OUT REQUEST kennzeichnet, daß auf dem Adreßbus eine Ein-/Ausgabe-Portadresse ansteht. IORQ wird außerdem im Interruptakzeptierungszyklus verwendet.
RD	READ Daten lesen
WR	WRITE Daten schreiben
M1	Befehlslesezykluskennzeichen (fetch-cycle)
BUSRQ	BUS REQUEST Mitteilung an den U 880, daß Adreß-Paten- und Steuersignalleitungen der ZVE entzogen werden sollen (z. B. für DMA)
BUSAK	BUS AKNOWLEDGE Antwort des U 880 auf BUSRQ Die angeforderten Signalleitungen sind frei
RFSH	REFRESH Auffrischungssignal für dynamische RAM-Speicher
RESET	Rücksetzen der ZVE (IFF1, IFF2, PC, I, R=0)
HALT	Stop der Befehlsabarbeitung und zyklische Ausführung intern erzeugter NOP-Instruktionen. Der U 880 ist nur durch Reset, NMI oder INT aus dem Halt zu befreien.

Signal-bezeichnung	Erläuterung
WAIT	CPU geht in einen Wartestand
INT	Interruptsignal
NMI	nicht maskierbares Interruptsignal
C	CLOCK Systemtakt (2,5 MHz)
U_{CC}	Betriebsspannung + 5 V
U_{SS} (GND)	Masse

2.2.1. Registerstruktur

Der Mikroprozessor U 880 hat die im Bild 2.3 dargestellte Registerstruktur mit einem zweifach vorhandenen Notizblockregister, bestehend aus Akkumulator, Flag-Register und sechs allgemeinen Registern. Der Zugriff ist immer nur zu dem gerade zur Arbeit spezifizierten Registersatz möglich. Nach dem Einschalten der ZVE ist das der Hauptregistersatz. Der Übergang zum Zweitregistersatz und zurück erfolgt über einen speziellen Befehl, die Exchangeanweisung. Durch diese alternative Registerstruktur wird eine sehr kurze Reaktionszeit bei zweistufig organisierten Interruptsystemen - Vordergrundprogramm/Hintergrundprogramm - ohne die Auslagerung von Registerinhalten in den RAM-Speicher erreicht. In Kleinstsystemen ohne RAM-Speicher kann der Zweitregistersatz außerdem als Operativ-Datenspeicher benutzt werden.
Ein 16-Bit-Befehlszähler PC (programm counter) beinhaltet bei der Befehlsabarbeitung den momentanen Befehlszählerstand. Die 16 Bits dieses Registers ermöglichen die direkte Adressierung von 65536 Speicherplätzen (64 k).
Der 16-Bit-Stack-Pointer dient zur Lokalisierung des Stack und zur Kennzeichnung des Fülletandes dieses Kellerspeichers für Unterprogrammrückkehradressen. Der Stack wird im Operativspeicherbereich (RAM) angeordnet. Die Anordnung des Stack und die Festlegung seiner Größe wird vom Programmierer über-

nommen. Dadurch wird eine formal unbegrenzte - real jedoch immer an die zu bearbeitende Aufgabe anzupassende - Verschachtelungstiefe von Unterprogrammen und Interruptserviceroutinen möglich.
Zwei 16-Bit-Indexregister erlauben zusammen mit einer im Befehlswort anzugebenden 8-Bit-Verschiebung die indizierte Adressierung in einem Bereich von 256 Byte. Sie sind außerdem als allgemeine 16-Bit-Register zu verwenden.
Das Interruptvektorregister wird zur Aufbewahrung der oberen acht Adreßbits verwendet, die zusammen mit dem vom Peripheriebaustein kommenden unteren acht Adreßbits den auf die entsprechende Serviceroutine zeigenden Interruptvektor ergeben.
Ein Memory-Refresh-Register dient zum gerätetechnisch unkomplizierten Anschluß dynamischer RAM-Speicherbauelemente.

Register für spezielle Zwecke

- Befehlszähler (PC)
  Der Befehlszähler enthält die 16-Bit-Adresse des aktuellen Befehls, der vom Speicher zu holen ist. Der Befehlszähler wird automatisch erhöht, nachdem sein Inhalt in die Adressenleitung überführt worden ist. Wenn Programmsprünge auftreten, wird der neue Wert automatisch in den PC überführt bei Überschreiten des erhöhten Wertes.

- Keller-Zeiger (SP = Stack pointer)
  Der Zeiger enthält die 16-Bit-Adresse des aktuellen obersten Wertes eines Kellers, der irgendwo in einem externen System-RAM untergebracht ist. Dieser externe Kellerspeicher ist als 'zuletzt hinein, zuerst heraus' (LIFO/-Datei) organisiert.
  Daten können durch die Ausführung der PUSH- oder POP-Befehle, von speziellen CPU-Registern aus gekellert werden oder aus dem Keller in bestimmte CPU-Register zurückgeholt werden. Die aus dem Keller geholten Daten sind immer

die, die als letzte zuvor gekellert wurden. Der Keller ermöglicht eine einfache Gestaltung von Mehrfach-Interrupts, unbegrenzter Interprogrammtechnik und Vereinfachungen bei vielen Arten von Datenbehandlungen.

- Zwei Indexregister (IX + IY)

Die zwei unabhängigen Register enthalten eine 16-Bit-Basis-Adresse, die bei indizierter Adressierung verwendet wird.
Dabei wird ein Indexregister als Basis benutzt, um das Gebiet im Speicher festzulegen, von dem aus Daten gespeichert oder entnommen werden sollen. Ein zusätzliches Byte ist in indizierten Befehlen enthalten, um die Entfernung von dieser Basis anzugeben. Diese Entfernung ist als Zweierkomplement der entsprechenden Zahl spezifiziert. Diese Adressierungsart vereinfacht in starkem Maße viele Typen von Programmen, speziell dort wo Tabellen-Daten benutzt werden.

- Interrupt-Vektor-Register (I)

Die CPU U 880 kann so betrieben werden, daß ein indirekter Aufruf zu irgendeinem Speicherplatz in Abhängigkeit von einem Interrupt erreicht werden kann. Das Register I enthält dann die höchsten 8 Bits der indirekten Adresse, während die den Interrupt auslösende Schaltung die unteren 8 Bits der Adresse liefert. Diese Verfahrensweise ermöglicht es, Interrupt-Routinen dynamisch irgendwo im Speicher mit absolut geringster Zugriffszeit zu dieser Routine abzuspeichern.

- Speicher-Auffrisch-Register (R)

Die CPU U 880 enthält einen Speicher-Auffrisch-Zähler, der dafür sorgt, daß dynamische Speicher genauso benutzt werden können wie statische. Dieses 7-Bit-Register wird

automatisch nach jedem Befehlsaufruf erhöht. Die Daten im Auffrisch-Zähler werden auf den unteren Teil des Adressenbusses mit einem Refresh-Steuersignal abgesetzt, während die CPU den aufgerufenen Befehl dekodiert und ausführt. Diese Art der Auffrischung ist vollständig programmtransparent und senkt nicht die Arbeitsgeschwindigkeit der CPU. Der Programmierer kann das Register R zu Testzwecken laden, aber das Register wird normalerweise nicht vom Programmierer benutzt.

Akkumulator und Flag-Register

Die CPU enthält zwei unabhängige 8-Bit-Akkumulatoren und verbunden damit zwei 8-Bit-Flag-Register. Der Akkumulator enthält die Ergebnisse der 8-Bit-Rechenoperationen oder logischen Operationen, wohingegen die Flag-Register die speziellen Bedingungen für die 8- oder 16-Bit-Operationen anzeigen, z. B., wenn das Ergebnis einer Operation gleich Null oder ungleich Null ist. Der Programmierer wählt das Akkumulator-Flag-Paar, mit dem er arbeiten möchte, durch einen einzigen Tausch-Befehl, so daß er mit jedem Paar arbeiten kann.

Allgebrauchs-Register

Es gibt zwei zueinander passende Sätze von Allgebrauchsregistern. Jeder Satz enthält sechs 8-Bit-Register, die einzeln als 8-Bit-Register oder paarweise als 16-Bit-Register durch den Programmierer verwendet werden können. Der eine Satz wird mit BC, DE und HL und der Alernativsatz mit BC' DE und HL bezeichnet.
Zu jeder beliebigen Zeit kann der Programmierer einen Satz mittels eines Austauschbefehls zur Benutzung auswählen.
In Systemen, wo schnelle Interruptbehandlung erforderlich ist, kann ein Satz von Allgebrauchsregister und ein Akkumulator mit Steuerflags für die Behandlung dieser schnellen Routinen reserviert werden. Nur ein einfacher Austausch-

befehl ist erforderlich, um die Anfangsbedingung der Routine zu laden. Das verkürzt wesentlich die Interruptbehandlungszeit. Diese Allgebrauchsregister sind in einem großen Anwendungsbereich durch den Programmierer zu nutzen. Sie vereinfachen auch die Programmierung, besonders bei auf ROM orientierten Systemen, bei denen nur ein kleiner externer Lese-/Schreibspeicherbereich verfügbar ist.

Rechenwerte und Logische Einheit (ALU)

Die arithmetischen und logischen 8-Bit-Befehle der CPU werden in der ALU ausgeführt. Intern steht die ALU mit dem Register und dem externen Datenbus in Verbindung. Die Funktionsarten, die durch die ALU ausgeführt werden, sind folgende:

Addition
Subtraktion
logisches UND
logisches ODER
logisches exklusiv ODER
Vergleichen
Links-/Rechtsverschiebung oder zyklische Verschiebung (arithmetisch und logisch)
Erhöhen um 1
Erniedrigen um 1
Bit-Setzen
Bit-Löschen
Bit-Testen

Befehlsregister und CPU-Steuerung

Wenn ein Befehl vom Speicher geholt worden ist, wird er ins Befehlsregister geladen und dekodiert. Das Steuerteil führt diese Funktion durch, erzeugt alle Signale, die erforderlich sind, um Daten von oder zu den Registern zu lesen oder zu schreiben, die ALU zu steuern, gibt diese Signale aus und

liefert alle extern erforderlichen Steuersignale.

2.2.2. Zeitverhalten der CPU U 880

Die CPU U 880 führt alle Befehle durch schrittweises Arbeiten eines genau festgelegten Satzes weniger Grundoperationen durch.
Diese umfassen:

Speicherlesen oder -schreiben
E/A-Gerätelesen oder -schreiben
Interruptaufnahme.

Alle Befehle bestehen aus einer Aneinanderreihung von diesen Grundoperationen. Jeder dieser Grundoperationen kann 3 bis 6 Takte dauern oder sie können durch zusätzliche WAIT-Zeiten verlängert werden, wenn die CPU mit der Geschwindigkeit externer Geräte synchronisiert werden muß. Die Taktperioden des Taktsignals sind als T-Zyklen und die Grundoperationen als M-Zyklen (für Maschinen -) bezeichnet.
Bild 2.4 zeigt, daß sich ein typischer Befehl nur aus einer Reihe von speziellen M- und T-Zyklen zusammensetzt. Man erkennt, daß dieser Befehl aus drei Maschinenzyklen (M1, M2, M3) besteht. Der erste Maschinenzyklus jedes Befehls ist der OP-Kode-Aufruf-Zyklus, der vier, fünf oder sechs T-Zyklen lang ist, (wenn er nicht durch ein WAIT-Signal verlängert wird). Im Aufrufzyklus (M1) wird der Operationskode des nächsten Befehls aufgerufen, um ihn danach auszuführen.
Die folgenden Maschinenzyklen transportieren die Daten zwischen der CPU und dem Speicher oder E/A-Geräten und können zwischen 3 und 5 T-Zyklen lang sein (sie können ebenfalls wieder durch ein WAIT-Signal verlängert werden, wenn die CPU mit externen Geräten zu synchronisieren ist).

In den folgenden Abschnitten werden die Zeitverläufe behandelt, die innerhalb eines beliebigen Grund-Maschinenzyklusses auftreten.

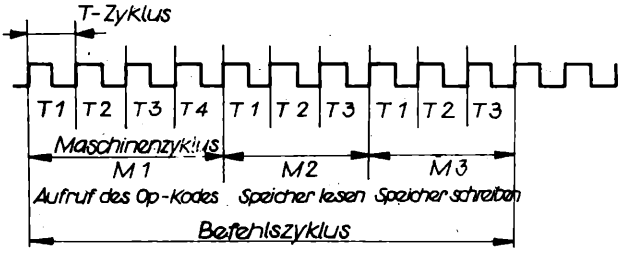


Bild 2.4: Beispiel des generellen Zeitablaufes in der CPU

Alle CPU-Zeitverläufe können in wenige sehr einfache Zeitabläufe unterteilt werden, die in den Bildern 2.5 bis 2.11 dargestellt sind.

Befehlsaufruf

Bild 2.5 zeigt die Zeitverläufe während eines M1-Zyklus (Aufruf des OP-Kodes). Zu beachten ist, daß der PC (Befehlszähler) zu Beginn des M1-Zyklus auf den Adressenbus gegeben wird. Eine halbe Taktzeit später wird das $\overline{\text{MREQ}}$-Signal aktiv. Bis zu diesem Zeitpunkt hatte die Adresse für den Speicher Zeit gehabt, um sich zu stabilisieren, so daß die Abfallflanke von $\overline{\text{MREQ}}$ direkt als Chip-Freigabetakt für die Speicher benutzt werden kann.
Die $\overline{\text{RD}}$-Leitung wird gleichzeitig dazu aktiv und zeigt damit an, daß die vom Speicher gelesenen Daten auf den CPU-Datenbus gegeben werden können. Die CPU übernimmt die Daten vom Datenbus mit der Anstiegsflanke des Taktes T3. Die gleiche Flanke wird durch die CPU auch benutzt, um die Signale $\overline{\text{RD}}$ und $\overline{\text{MREQ}}$ abzuschalten.
Folglich sind die Daten schon durch die CPU übernommen, bevor das RD-Signal inaktiv wird. Die Takte T3 und T4 des

Befehlsaufrufzyklus werden benutzt, um die dynamischen Speicher aufzufrischen. In der CPU wird diese Zeit benutzt, um den aufgerufenen Befehl zu dekodieren und auszuführen. Während T3 und T4 enthalten die unteren 7 Bits des Adressenbusses die Speicheradresse für die Auffrischung und das $\overline{RFSH}$-Signal wird aktiv und zeigt damit an, daß ein Auffrischlesen aller dynamischen Speicher durchgeführt werden kann. Ein $\overline{RD}$-Signal wird während der Auffrischzeit nicht erzeugt, um zu vermeiden, daß Daten von verschiedenen Speicherbereichen auf den Datenbus gegeben werden. Das $\overline{MREQ}$-Signal kann dazu benutzt werden, während der Auffrischzeit ein Auffrischungslesen aller Speicherelemente durchzuführen. Das Auffrischsignal selbst kann dazu nicht verwendet werden, weil die Auffrischadresse nur während der $\overline{MREQ}$-Zeit garantiert stabil ist.

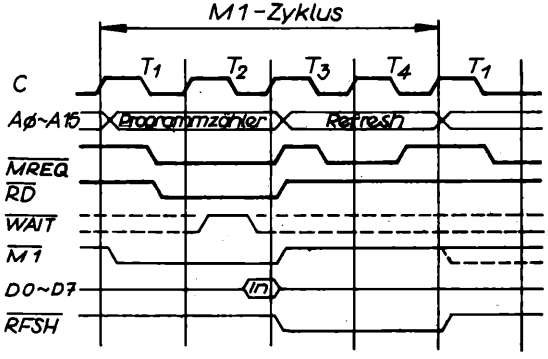


Bild 2.5 Zeitverhalten OP-Code Aufruf-Zyklus (M1-Zyklus)

Der Befehlsaufrufzyklus wird verzögert, wenn der Speicher die $\overline{WAIT}$-Leitung aktiviert. In T2 und jedem nachfolgenden TW tastet die CPU die $\overline{WAIT}$-Leitung während der Abfallflanke des Taktes ab. Wenn die $\overline{WAIT}$-Leitung während dieser Zeit aktiv ist, wird ein weiterer $\overline{WAIT}$-Zustand im folgenden T-Zyklus erzeugt.

Mit diesem Verfahren kann der Lesezyklus beliebig verlängert und so an die Zugriffszeit von beliebigen Speicheranordnungen angepaßt werden.

Speicherlesen oder -schreiben

Bild 2.6 stellt die Zeitverläufe während der Speicherlese- oder Schreibzyklen dar, die sich vom M1-Zyklus unterscheiden. Diese Zyklen sind allgemein 3 Taktperioden lang, wenn nicht über das $\overline{WAIT}$-Signal durch den Speicher WAIT-Zustände erzwungen werden.
Das $\overline{MREQ}$- und das $\overline{RD}$-Signal werden genauso verwendet wie im Aufrufzyklus. Im Falle des Speicherschreibzyklus wird $\overline{MREQ}$ aktiv, wenn der Adressenbus stabil ist, so daß es direkt als Chip-Freigabe für Speicher verwendet werden kann. Die $\overline{WR}$-Leitung ist aktiv, wenn die Daten auf dem Datenbus stabil sind, so daß dieses Signal direkt als ein R/W-Impuls für jede Type von Halbleiterspeichern verwendet werden kann. Das WR-Signal wird einen halben Takt, bevor die Adressenbus- und die Datenbus-Inhalte verändert werden, inaktiv, so daß die Überlappungserfordernisse im Prinzip für alle Typen von Halbleiterspeichern erfüllt werden.

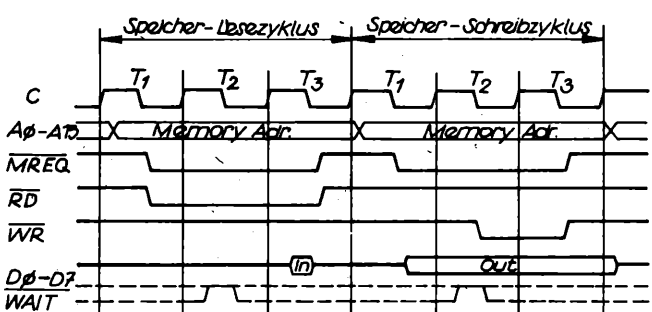


Bild 2.6: Zeitverhalten Speicher-Lesezyklus und Speicher-Schreibzyklus

Ein- und Ausgabe-Zyklen

Bild 2.7 zeigt die E/A-Lese- oder E/A-Schreiboperation. Zu beachten ist, daß während der E/A-Operation automatisch ein WAIT-Zustand eingebaut wird. Der Grund dafür ist, daß bei E/A-Operationen die Zeit vom Aktivwerden des $\overline{IORQ}$-Signals bis zum Zeitpunkt, zu dem die CPU die WAIT-Leitung abfragt, sehr kurz ist und damit ohne diesen extra WAIT-Zustand für die E/A-Kanäle nicht genügend Zeit zur Verfügung steht, um die Adresse zu dekodieren und gegebenenfalls die WAIT-Leitung zu aktivieren. Während dieses WAIT-Zustandes wird das $\overline{WAIT}$-Forderungssignal abgetastet. Während einer Lese-E/A-Operation wird die $\overline{RD}$-Leitung benutzt, um den adressierten Kanal auf den Datenbus zu schalten, wie es in dem Falle des Speicher-Lesens der Fall war. Bei E/A-Operationen wird die $\overline{WR}$-Leitung als Takt für die E/A-Kanäle benutzt, wiederum mit automatisch zur Verfügung stehender ausreichender Überlappungszeit, so daß die Anstiegsflanke als Daten-Takt benutzt werden kann.

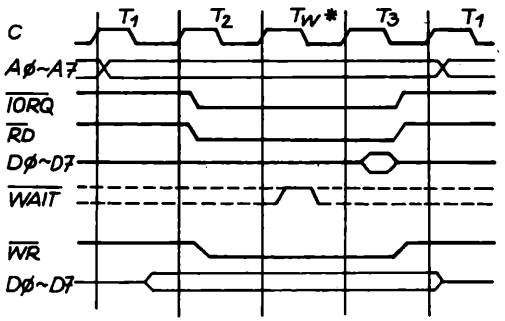


Bild 2.7: Zeitverhalten E/A-Lesezyklus und E/A-Schreibzyklus

Bus-Anforderungs-/Bus-Bestätigungszyklus

Im Bild 2.8 ist das Zeitverhalten für einen Busanforderungs-/Busbestätigungszyklus dargestellt. Das $\overline{BUSRQ}$ -Signal wird von der CPU mit der Anstiegsflanke des letzten Taktes eines jeden Maschinenzyklus abgetastet. Wenn das $\overline{BUSRQ}$-Signal aktiv ist, setzt bei der Anstiegsflanke des nächsten Taktimpulses die CPU ihre Adressen-, Daten- und Drei-Zustands-Steuersignale in den hochohmigen Zustand. Zu dieser Zeit kann jedes externe Gerät die Busse steuern und die Daten zwischen Speicher und E/A-Geräten transportieren. Dies wird allgemein als direkter Speicherzugriff bezeichnet (DMA-direct memory access). Die maximale Zeitdauer bis zur Antwort der CPU auf eine Busanforderung ist die Länge des Maschinenzyklus. Die externe Steuerung kann die Bus-Steuerung über so viele Takte behalten, wie es erforderlich ist. Zu beachten ist, daß bei sehr langen DMA-Zyklen und bei Verwendung von dynamischen Speichern die Auffrischfunktion durch die externe Steuerung übernommen werden muß.
Diese Situation tritt nur auf, wenn sehr große Blöcke von Daten mit DMA-Steuerung übertragen werden. Zu beachten ist weiterhin, daß die CPU während des Busanforderungszyklus weder durch $\overline{NMI}$ noch durch ein $\overline{INT}$-Signal unterbrochen werden kann.

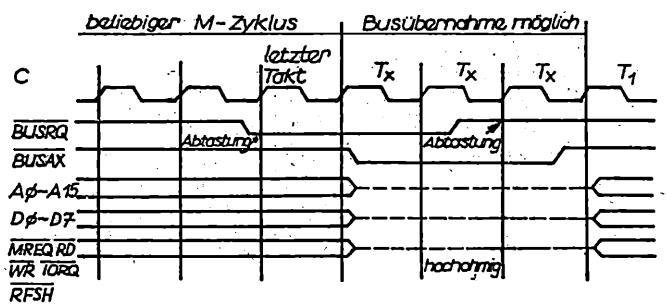
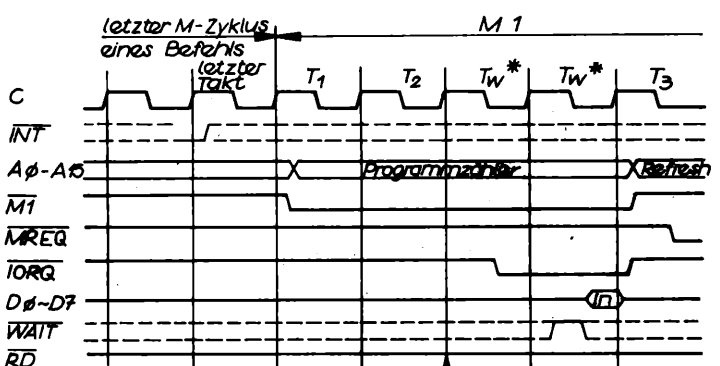


Bild 2.8 Zeitverhalten beim Busanforderungs-/Bestätigungszyklus.

Interrupt-Anforderungs-/Annahme-Zyklus

Bild 2.9 zeigt das Zeitverhalten bei einem Interrupt-Zyklus. Das Interrupt-Signal ($\overline{INT}$) wird durch die CPU an der Anstiegsflanke des letzten Taktes am Ende eines Befehls abgefragt. Das Signal wird nicht berücksichtigt, wenn das Interrupt-Annahme-Flip-Flop, das durch die interne CPU-Software gesteuert wird, nicht eingeschaltet ist oder, wenn das $\overline{BUSRQ}$-Signal aktiv ist. Wenn das Signal angenommen wird, wird ein spezieller M1-Zyklus erzeugt. Während dieses speziellen M1-Zyklus wird das $\overline{IORQ}$-Signal aktiv (im Gegensatz zum Normalfall mit $\overline{MREQ}$), um zu melden, daß das die Unterbrechung anfordernde Gerät seinen 8-Bit-Vektor auf den Datenbus geben kann. In diesen Zyklus werden automatisch zwei Waitzustände aufgenommen. Diese Takte wurden eingefügt, um eine Interruptanordnung mit gestufter Priorität leichter verwirklichen zu können. Diese zwei WAIT-Zustände geben den Signalen genügend Zeit, um sich zu stabilisieren und festzustellen, welches E/A-Gerät in den Antwortvektor eingesetzt werden muß.



2.9 Zeitverhalten beim Interruptanforderungs-/Annahmezyklus

Nicht maskierbares Interrupt-Verhalten

Bild 2.10 zeigt den Anforderungs-/Annahme-Zyklus für einen nicht maskierbaren Interrupt. Das Signal wird zur gleichen Zeit wie die Interrupt-Leitung abgefragt, aber diese Leitung hat eine höhere Priorität als der normale Interrupt und kann durch eine Software-Steuerung nicht abgewiesen werden. Seine Funktion besteht darin, sofort eine Antwort auf wichtige Signale (z. B. drohender Netzausfall) zu liefern. Die CPU-Antwort auf einen nicht maskierbaren Interrupt ist ähnlich einer normalen Speicherlese-Operation. Der einzige Unterschied besteht darin, daß der Inhalt des Datenbusses ignoriert wird, weil die CPU automatisch den Befehlszählerstand im externen Keller speichert und zur Adresse 0066_H springt. Die Routine zur Behandlung eines nicht maskierbaren Interrupts muß an diesem Speicherplatz beginnen, wenn diese Interruptart verwendet werden soll.

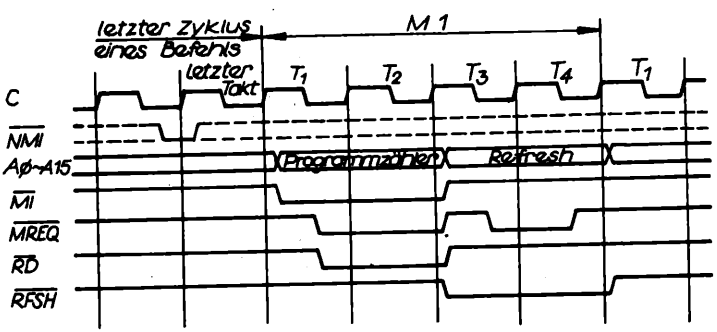


Bild 2.10 Zeitverhalten beim NMI-Anforderungs-/Annahmezyklus

HALT-Ausführung

Wenn ein Software-Halt-Befehl abzuarbeiten ist, beginnt die CPU Leerbefehle (NOP's) auszuführen, bis ein Interrupt empfangen wird (entweder ein nicht maskierbarer oder ein maskierbarer Interrupt, wenn das Interrupt-Annahme-Flip-Flop eingeschaltet ist). Die zwei Interrupt-Leitungen werden an jeder Anstiegsflanke des T4-Taktes abgefragt, wie das im Bild 2.11 gezeigt wird. Wenn ein nicht maskierbarer Interrupt oder ein maskierbarer Interrupt bei eingeschaltetem Interrupt-Annahme-Flip-Flop empfangen wird, dann wird der HALT-Zustand bei der nächsten Anstiegsflanke noch ausgeführt. Der folgende Zyklus wird dann der Interrupt-Annahme-Zyklus entsprechend dem Typ des empfangenen Interrupts. Wenn zu dieser Zeit beide empfangen wurden, dann wird der nicht maskierbare Interrupt angenommen, weil er die höchste Priorität besitzt. Die Leerbefehle (NOP's) werden während des HALT-Zustandes ausgeführt, um die Speicher-Auffrisch-Signale aktiv zu halten. Jeder Zyklus im HALT-Zustand ist ein normaler M1-(Aufruf)Zyklus mit der Ausnahme, daß die vom Speicher empfangenen Daten ignoriert werden und daß die Leerbefehle intern in der CPU erzeugt werden. Das $\overline{HALT}$-Annahmesignal ist während dieser Zeit aktiv, um zu melden, daß sich die CPU im HALT-Zustand befindet.

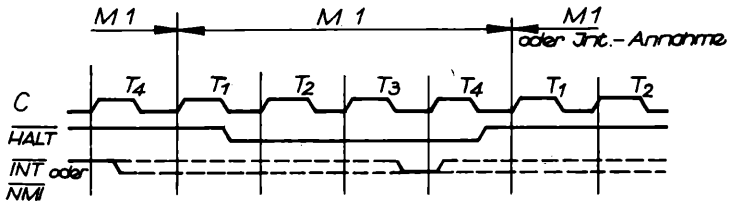


Bild 2.11 Zeitverhalten bei HALT-Ausführung

Zeitverhalten mit WAIT-Zuständen

Das Zeitverhalten beim M1-Zyklus, beim Speicher-Lese-oder-Schreibzyklus und beim Interruptanforderungs-/Annahmezyklus kann durch das $\overline{WAIT}$-Signal gezielt verzögert werden. In den genannten Zyklen wird jeweils im Taktzustand T2 und in allen eventuell darauf folgenden Zuständen T_W die $\overline{WAIT}$-Leitung während der Abfallflanke des Taktes abgetastet. Wenn die $\overline{WAIT}$-Leitung während der Abtastzeit aktiv ist, wird ein weiterer WAIT-Zustand im folgenden Takt erzeugt. Durch das Einfügen von solchen zusätzlichen WAIT-Zuständen kann das Zeitverhalten der CPU an beliebige Speicher oder E/A-Schaltungen angepaßt werden.

2.2.3. Interruptsystem

Der Zweck eines Interrupts besteht darin, es den peripheren Geräten zu ermöglichen, die CPU-Operation in einer sinnvollen Weise zu unterbrechen und die CPU zu zwingen, daß eine Routine zur Bedienung der Peripherie gestartet wird. Gewöhnlich sind in diesen Routinen Austauschoperationen für Daten, Status- oder Steuerinformationen zwischen der CPU und der Peripherie eingeschlossen. Wenn die Bedienungsroutine abgearbeitet ist, kehrt die CPU zu der Operation zurück, bei der sie unterbrochen wurde.

Interrupt-Annahme/-Abweisung

Die CPU U 880 hat zwei Interrupteingänge, einen durch die Software maskierbaren ($\overline{INT}$) und einen nicht maskierbaren Interrupt ($\overline{NMI}$). Die Annahme des nicht maskierbaren Interrupts kann durch den Programmierer nicht verhindert werden. Er wird immer angenommen, wenn ein peripheres Gerät ihn fordert. Dieser Interrupt wird i.a. für die wichtigsten Funktionen reserviert, die beim Auftreten sofort bedient werden müssen, z. B. ein bevorstehender Stromausfall.

Neben den zwei Interrupteingängen besitzt die CPU noch den BUSRQ-Steuereingang für die Busübergabe, der speziell bei DMA-Betrieb Verwendung findet. Dieser Steuereingang hat gegenüber allen Interruptanforderungen höchste Priorität. Insgesamt ergibt sich in der CPU folgende Prioritätsbewertung:

1. Priorität: Bus-Request ($\overline{BUSRQ}$)
2. " nicht maskierbarer Interrupt ($\overline{NMI}$)
3. " maskierbarer Interrupt ($\overline{INT}$)

Der maskierbare Interrupt kann durch den Programmierer selektiv zugelassen oder abgewiesen werden. Damit hat der Programmierer die Möglichkeit, Interrupts abzuweisen, wenn während bestimmter Perioden das Programm ein Verhalten realisieren muß, in dem ein Interrupt nicht zulässig ist.
In der CPU U 880 gibt es ein Annahme-Flip-Flop (IFF1), das durch den Programmierer mit den Befehlen Interrupt-Annehmen (Enable interrupt-EI) bzw. Interrupt-Abweisen (Disable interrupt - DI) ein - bzw. ausgeschaltet werden kann. Wenn IFF1 ausgeschaltet ist, kann durch die CPU kein Interrupt angenommen werden.

Tatsächlich gibt es in der CPU U 880 2 Annahme-Flip-Flops: IFF1 und IFF2.

IFF1	IFF2
Verhindert aktuell die Annahme von Interrupts	zeitweiliger Speicherplatz für IFF1

Die Stellung von IFF1 wird benutzt, um aktuelle Interrupts abzuweisen, während IFF2 nur als zeitweiliger Speicherplatz für IFF1 dient. Der Zweck, IFF1 zu speichern, ist folgender:

Ein RESET der CPU schaltet u. a. IFF1 und IFF2 aus, so daß Interrupts abgewiesen werden. Sie können zu beliebiger Zeit durch den Programmierer mit dem EI-Befehl zugelassen werden. Wenn ein EI-Befehl ausgeführt ist, wird eine schon anliegende Interruptanforderung erst nach der Abarbeitung des auf EI folgenden Befehls angenommen. Diese Verzögerung um einen Befehl ist für den Fall erforderlich, wenn der auf EI folgende ein Rücksprung (return) ist, da ein Interrupt nicht zugelassen werden kann, bis der Rücksprung abgearbeitet ist.
Der EI-Befehl setzt sowohl IFF1 als auch IFF2 in den Annahme-Zustand. Wenn ein Interrupt durch die CPU angenommen wird, werden sowohl IFF1 als auch IFF2 automatisch ausgeschaltet, um weitere Interrupts zu verhindern, bis der Programmierer einen neuen EI-Befehl benutzt. D. h. in den oben genannten Fällen sind IFF1 und IFF2 immer gleich.

Der Zweck, in IFF2 den Zustand von IFF1 zu speichern, wird deutlich, wenn ein nichtmaskierbarer Interrupt auftritt. Wenn ein nicht maskierbarer Interrupt angenommen wird, wird IFF1 ausgeschaltet, um weitere Interrupts zu verhindern, bis der Programmierer sie wieder zulassen will. Folglich werden, nachdem ein nicht maskierbarer Interrupt angenommen wurde, maskierbare Interrupts abgewiesen, aber der vorherige Zustand von IFF1 ist gerettet worden, so daß der komplette Zustand der CPU, wie er vor dem nicht maskierbaren Interrupt bestanden hatte, wieder hergestellt werden kann. Wenn der Befehl "Laden des Akkumulators vom Register I" (LDA,I) oder der Befehl "Laden des Akkumulators vom Register R" (LD A,R) ausgeführt ist, ist der Zustand von IFF2 in das Parität-Flag überführt worden, wo er getestet oder gespeichert werden kann.

Eine zweite Methode, den Zustand von IFF1 zurückzugewinnen, ist die Abarbeitung des Befehls "Rücksprung vom nicht maskierbaren Interrupt" ($\overline{\text{RETN}}$). Da dieser Befehl anzeigt, daß die Behandlungsroutine eines nicht maskierbaren Interrupts abgearbeitet ist, wird der Inhalt von IFF2 nach IFF1 überführt, so daß der Zustand von IFF1 automatisch so wiederhergestellt ist, wie er vor der Annahme des nicht maskierbaren Interrupts bestanden hatte.

Nachfolgend ist die Wirkung verschiedener Befehle auf die zwei Interrupt-Annahme-Flip-Flops zusammengestellt:

AKTION	IFF1	IFF2	
RESET	0	0	
DI	0	0	
EI	I	I	
LD A,I	.	.	IFF2 Paritäts-Flag
LD A,R	.	.	IFF2 "
Annahme von NMI	0	.	
RETN	IFF2	.	IFF2 IFF1

"." bedeutet keine Veränderung

Bild 2.12 zeigt den logischen Ablauf der Bewertung der drei Steuereingänge $\overline{BUSRQ}$, $\overline{NMI}$ und $\overline{INT}$. Die $\overline{BUSRQ}$-Leitung wird zuerst getestet. Das interne BUSRQ-Flip-Flop (BUSRQ-F/F) wird gesetzt, wenn die $\overline{BUSRQ}$-Leitung bei der Anstiegsflanke des letzten Taktes jedes Maschinenzyklus aktiv ist. Ist dieser Maschinenzyklus gleichzeitig der letzte Maschinenzyklus der in Ausführung befindlichen Anweisung, so ist die CPU auch zur Übernahme der nicht maskierbaren oder maskierbaren Interruptsignale bereit. In diesem Fall werden nach der $\overline{BUSRQ}$-Leitung die $\overline{NMI}$-Leitung bzw. die $\overline{INT}$-Leitung getestet. Da die $\overline{NMI}$-Leitung flankengetriggert ist und ihre negativen Flanken ein $\overline{NMI}$-Eingangs-Flip-Flop (NMI-EFF) setzen, wird beim Test der $\overline{NMI}$-Leitung eingentlich das $\overline{NMI}$-Eingangs-Flip-Flop abgefragt und ausgewertet.
Bei einem gesetzten NMI-Eingangs-Flip-Flop werden dann das interne NMI-Flip-Flop gesetzt und das INT-Annahme-Flip-Flop IFF1 rückgesetzt. Wenn das NMI-Eingangs-Flip-Flop seinerseits nicht gesetzt ist, prüft die CPU nun den Zustand der $\overline{INT}$-Leitung und setzt bei einer aktiven $\overline{INT}$-Leitung und einer nicht aktiven Interruptverhinderung (IFF1 = 0) das interne INT-Flip-Flop.
Die Abarbeitung erfolgt dann in der Reihenfolge BUSRQ-F/F und INT-F/F.

In der Tafel 2.2 sind nochmals alle Reaktionen der CPU auf die drei Steuerleitungen $\overline{BUSRQ}$, $\overline{NMI}$ und $\overline{INT}$ unter Berücksichtigung aller Randbedingungen aufgelistet. Die steigende Flanke des letzten Taktes (T_L) innerhalb eines Maschinenzyklus ist immer für die Übernahme der Steuerinformation in die internen Flip-Flops verantwortlich. Im Falle des Maschinenzyklus $\overline{MI}$ ist $T_L = T_4$, im Falle von Speicher-Schreib/Lesezyklen ist $T_L = T_3$. Bei der Priorisierung von $\overline{NMI}$ gegenüber $\overline{INT}$ ist zu beachten, daß NMI nur dann vorgezogen wird, wenn die Anforderung auch wirklich vor der steigenden Flanke des letzten Taktes T_L des entsprechenden Befehls erscheint. In diesem Fall wird $\overline{INT}$ nicht in das interne INT-F/F übernommen, also bis zum Ende der NMI-Service-Routine ignoriert. Ansonsten erfolgt die Ausführung des ersten Befehls im Service-Programm für $\overline{INT}$ und erst dann der Sprung zur $\overline{NMI}$-Service-Routine.

Tafel 2.2

Signal	Auftreten des Signals	Obernahme des Signals	Wirkung	Bemerkung
$\overline{BUSRQ}$		nur bei steigender Flanke eines jeden T_L	Bestätigung (BUSAK) erscheint sofort vor dem nächsten Maschinenzyklus	$\overline{BUSRQ}$ hat Priorität vor $\overline{NMI}$ und $\overline{INT}$
$\overline{NMI}$	innerhalb eines Befehlszyklus vor der steigenden Flanke des nächsten T_L	wird jederzeit im NMI-Eingange-Flip-Flop abgespeichert (80 ns mind.)	sofort nach Abarbeitung des Befehls Sprung zur NMI-Service-Routine (auch unmittelbar nach einem EI-Befehl)	zur wiederholten Ausführung einer NMI-Service-Routine muß das NMI-Sign. kurzzeitig aktiv werden
	innerhalb eines Befehlszyklus nach der steigenden Flanke d. nächsten T_L	- " -	Abarbeitung d.laufenden und des nächstfolgenden Befehls, erst dann Sprung zur NMI-Service-Routine	
$\overline{INT}$		nur bei steigender Flanke des letzten T_L eines jeden Befehls und nur dann, wenn Interrupt freigegeben ist. Andernfalls wird $\overline{INT}$ nicht akzeptiert und ignoriert		Ausnahme: Während der Ausführung eines EI-Befehls wird $\overline{INT}$ nicht akzeptiert unabhängig vom vorherigen Zustand von IFF1 und IFF2

Interrupt-Beantwortung

CPU

- nicht maskierbar

Ein nicht maskierbarer Interrupt wird durch die CPU zu jeder Zeit angenommen. Wenn dieser auftritt, ignoriert die CPU den nächsten aufzurufenden Befehl und führt dafür einen RESTART zur Adresse 0066H durch. Folglich verhält sie sich genauso, als hätte sie einen RESTART-Befehl aufgerufen, mit dem Unterschied, daß diese Adresse keine der 8 Software-RESTART-Adressen ist.
Ein RESTART ist ein Unterprogrammaufruf von einer bestimmten Adresse im Anfangsbereich des Speichers.

- maskierbar

Die CPU kann so programmiert werden, daß sie auf einen maskierbaren Interrupt in einer der drei möglichen Arten (Mode) antwortet.

Mode 0

Dieser Mode ist identisch dem Interruptverhalten der CPU 8080A.

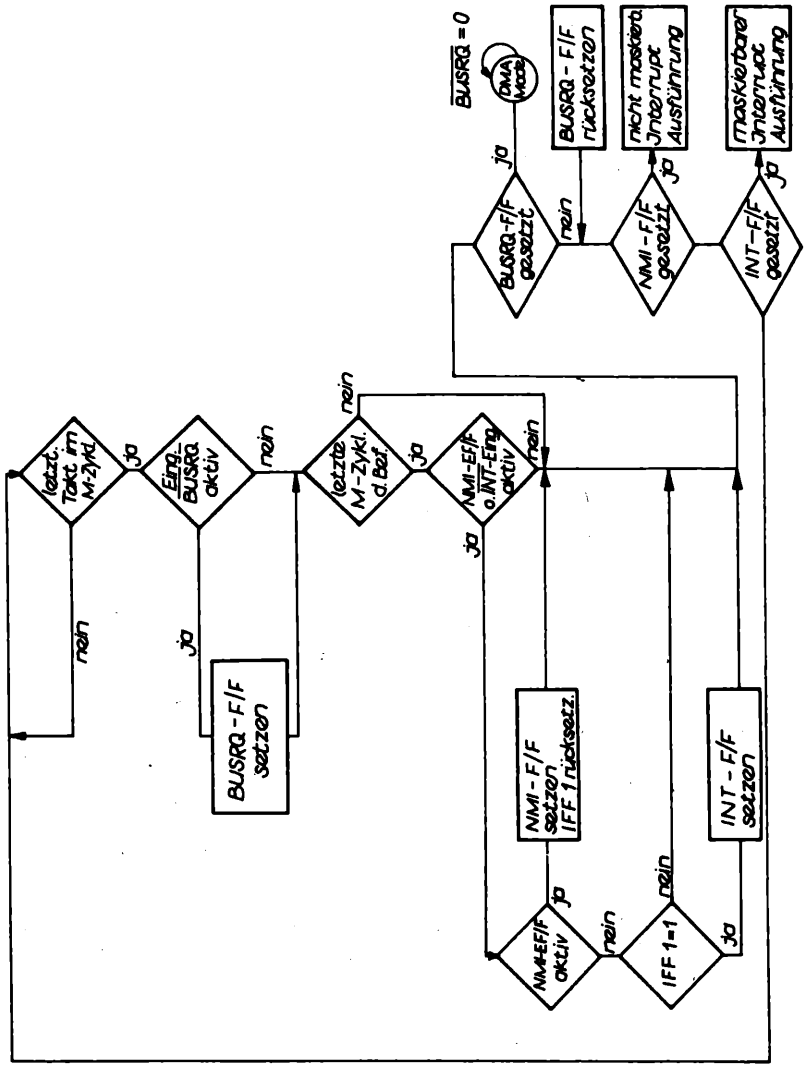


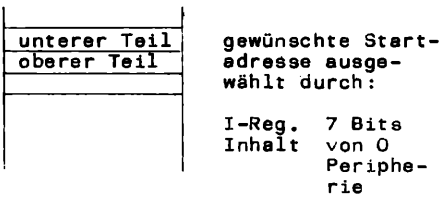
Bild 2.12 Logischer Ablauf der Bewertung von $\overline{BUSRQ}$, $\overline{NMI}$ und $\overline{INT}$

Bei diesem Mode kann die den Interrupt anfordernde Schaltung einen Befehl auf den Datenbus ausgeben und die CPU wird ihn ausführen. Folglich liefert die den Interrupt anfordernde Schaltung den nächsten abzuarbeitenden Befehl anstelle des Speichers. Meist wird das ein RESTART-Befehl sein, weil die den Interrupt anfordernde Schaltung nur einen Ein-Byte-Befehl einspeisen kann. Mit anderen Worten, es kann kein beliebiger anderer Befehl, wie z. B. ein 3-Byte-Unterprogrammruf, ausgeführt werden. Die Taktzahl für die Ausführung dieses Befehls ist um 2 Takte größer als die normale Zahl für diesen Befehl. Das kommt daher, weil die CPU automatisch 2 WAIT-Zustände in den Interrupt-Antwortzyklus einfügt, um der externen Logikkette (daisy chain) genügend Zeit für die Prioritätssteuerung zur Verfügung zu stellen.
Nach einem RESET ist die CPU immer automatisch auf Mode 0 gesetzt.

Mode 1
Wenn dieser Mode durch den Programmierer ausgewählt ist, wird die CPU einen Interrupt mit einem RESTART 0038H beantworten. Folglich ist diese Anwort mit der auf einen nicht maskierbaren Interrupt identisch mit der Ausnahme, daß jetzt die Adresse 0038H anstelle von 0066H aufgerufen wird. Ein weiterer Unterschied besteht darin, daß die erforderliche Zyklenzahl um 2 gegenüber dem normalen RESTART entsprechend den zwei eingefügten WAIT-Zuständen vergrößert ist.

Mode 2
Dieser Mode ist die leistungsfähigste der Interrupt-Antwort-Varianten. Mit einem einzigen 8-Bit-Byte kann vom Benutzer ein indirekter Unterprogrammaufruf zu einem beliebigen Speicherplatz durchgeführt werden. Bei diesem Mode stellt der Programmierer eine Tabelle mit 16-Bit-Startadressen für jede Interrupt-Behandlungsroutine auf. Diese Tabelle kann irgend-

wo im Speicher untergebracht sein. Wenn ein Interrupt angenommen wird, muß ein 16-Bit-Zeiger gebildet werden, um die Startadresse der gewünschten Interrupt-Behandlungsroutine aus der Tabelle holen zu können. Die oberen 8 Bit des Zeigers werden aus dem Inhalt des I-Registers gebildet. Das Register I muß zuvor mit dem vom Programmierer gewünschten Wert geladen werden, d. h. LD I,A. Zu beachten ist, daß ein RESET der CPU auch das Register I zurückstellt, das heißt, daß dort eine Null eingeschrieben wird. Die unteren 8 Bit müssen von der den Interrupt anfordernden Schaltung geliefert werden, wobei das niederwertigste Bit eine 0 sein muß. Das ist deshalb notwendig, weil der 16-Bit-Zeiger dazu verwendet wird, zwei aufeinanderfolgende Bytes aus einer Interrupttabelle zu holen, um die vollständige 16-Bit-Startadresse der Behandlungsroutine zu bilden. Die Startadressen der Interrupt-Routinen müssen in der Tabelle immer an geraden Speicherplätzen beginnen.



Das erste Byte jeder Adresse in dieser Tabelle ist der niederwertige Teil der Adresse. Der Programmierer muß selbstverständlich diese Tabelle mit den gewünschten Adressen füllen, bevor ein Interrupt angenommen werden darf.
Zu beachten ist, daß diese Tabelle zu jeder Zeit durch den Programmierer verändert werden kann, wenn verschiedene periphere Schaltungen mit verschiedenen Behandlungsroutinen bedient werden sollen. Voraussetzung dafür ist, daß die Tabelle in einem Lese-Schreib-Speicher untergebracht ist.

Wenn eine den Interrupt anfordernde Schaltung den unteren Teil des Zeigers liefert, kellert die CPU den Befehlsstand automatisch. Sie holt dann die Startadresse aus der Tabelle und führt einen Sprung zu dieser Adresse aus. Dieser Antwort-Mode benötigt 19-Takt-Perioden (7, um die unteren 8 Bit von der den Interrupt anfordernden Schaltung aufzurufen, 6, um den Befehlszähler zu retten, und 6, um die Sprungadresse zu bilden).

Zu beachten ist, daß die peripheren Schaltungen des U 880-Systems für die Interrupt-Prioritäten eine Logikkettenstruktur (daisy chain structure) aufweisen, die der CPU während d er Interrupt-Annahme automatisch einen programmierten Vektor liefert. Nähere Informationen dazu können aus dem Abschnitt über die Peripherieschaltkreise entnommen werden.

2.3. Wiederholungsfragen

1. Aus welchen Grundbausteinen besteht das Mikroprozessorsystem U 880? Geben Sie eine kurze Erläuterung dieser Schaltkreise!

2. Welche Speichertypen benötigt man zum Aufbau von Mikrorechnersystemen? Worin unterscheiden sie sich?

3. Welche Register besitzt die CPU U 880? Wozu dienen sie?

4. Worin unterscheiden sich die Interruptbetriebsarten bei der CPU U 880?

5. Wie ist der grundsätzliche Ablauf in der Interruptbetriebsart Mode 2?

3. Lernsystem

3.1. Beschreibung der Konfiguration

3.1.1. Allgemeines

Das Lernsystem stellt eine Mikrorechner Minimalkonfiguration dar, mit dem der Anwender in die Lage versetzt wird, das Mikroprozessorsystem U 880 sowie dessen Befehlssatz kennen zu lernen. Weiterhin ermöglicht es, kleinere Programme zu erarbeiten und zu testen. Durch die vielfältigen Funktionen ist das Lernsystem universell einsetzbar. Bild 3.1 zeigt das Blockschaltbild.

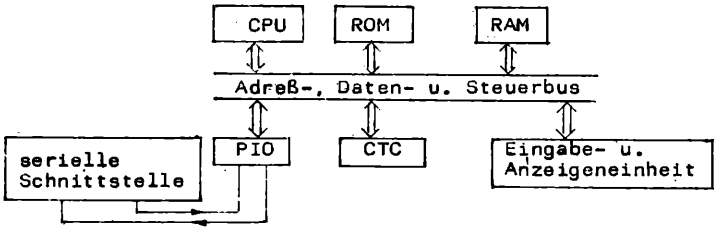


Bild 3.1

3.1.2. Zentraleinheit

Die Bausteine CPU, PIC und CTC befinden sich auf der Leiterkarte CPU1. Diese Baugruppe enthält weiterhin die für das System notwendige Takterzeugung, Power-on-reset-Schaltung, Treiberstufen und eine serielle Schnittstelle. Die Bausteine PIO und CTC stehen dem Anwender zur freien Verfügung, wobei der CTC in der Interruptkette die höchste Priorität besitzt. Technische Erläuterungen hierzu erfolgen unter Pkt. 2. und Pkt. 5.

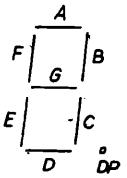
3.1.3. Speichereinheit

Die Baugruppe RR2 enthält alle notwendigen Speicher sowie den Adreßdekoder. Der ROM-Bereich umfaßt 8 kByte. Das Betriebsprogramm befindet sich in zwei EPROM-Schaltkreisen U 555 C. Als Arbeitsspeicher werden 8 Schaltkreise U 202 D (1024 x 1 Bit statisches RAM) verwendet, so daß 1 kByte RAM-Kapazität mit dem Adreßbereich F000H...F3FFH zur Verfügung stehen. Das Betriebsprogramm des Lernsystems belegt den RAM-Bereich 0F381H...0F3FFH. Für den Anwender ist somit der RAM-Bereich 0F000H...0F380H nutzbar.

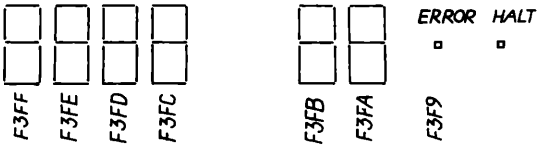
3.1.4. Eingabe- und Anzeigeeinheit

Die Eingabe- und Anzeigeeinheit besteht aus einer Tastatur und einer 7-Segment-Anzeige.
Alle Funktionen werden mittels der Tastatur ausgelöst. Diese ist als Matrix bestehend aus 32 Tasten ausgeführt (4 Zeilen zu je 8 Tasten). Die Reset-Taste wirkt sofort auf das System. Die anderen 31 Tasten werden softwaremäßig ausgewertet. Das Betriebsprogramm organisiert die Ansteuerung der Tastenhardware, das Einlesen, die Entprellung, die Signifikanzkontrolle sowie die Auswertung der Tasteneingaben.
Die Anzeige von Adressen und Daten erfolgt über ein 6-stelliges 7-Segment-Display, welches in eine Gruppe von 4 und eine Gruppe von 2 Stellen aufgetrennt ist. Über zwei Einzel-LEDs werden die Informationen "Halt" und "Error" zur Anzeige gebracht.
Die Zuordnung der Datenbits zu den einzelnen Segmenten der Anzeige ist wie folgt festgelegt:

D0 - Segment A
D1 - Segment B
D6 - Segment G
D7 - Dezimalpunkt



Die anzuzeigende Information wird im 7-Segment-Kode in entsprechend zugeordneten RAM-Speicherplätzen abgelegt, wobei folgende Zuordnung gilt:



Das Anzeigeprogramm steuert die einzelnen Stellen direkt entsprechend den Speicherinhalten an, d. h. ist Bit 0 = 1, dann leuchtet Segment A usw. Bei diesem Organisationsprinzip besteht die Möglichkeit, sowohl Daten im 7-Segment-Kode als auch beliebige andere Kodierungen in den Speicherzellen abzulegen und anzuzeigen.
Die Ansteuerung der Anzeige erfolgt zeitmuliplex, d. h. die einzelnen Stellen werden zeitlich nacheinander angesteuert. Organisiert wird die Anzeigebedienung durch NMI-Interrupts. Die Speicherinhalte bleiben dabei unverändert.
Hierzu ein kleines Beispiel: Bedienen Sie das Lernsystem wie folgt:

```
SET PC F000 EX
INP
21 EX
FA EX
F3 EX
36 EX
01 EX EX
SET PC F000 EX
SET SP F200 EX
STEP
STEP
```

In der rechten Anzeige erlischen alle Segmente außer dem Segment A, da der Speicherplatz F3FAH mit 01 geladen wurde.

3.2. Monitor-Kommandos

3.2.1. Allgemeine Hinweise

Die Monitor-Kommandos werden über das Funktionstastenfeld, die dazugehörenden Adressen bzw. Daten über das Daten- und Adreßfeld eingegeben. Alle Register, deren Inhalt per Tastatur angezeigt bzw. beeinflußt wird, sind Register einer virtuellen CPU, d. h. diese Werte befinden sich in dafür reservierten RAM-Zellen. Die CPU-Register werden erst zu Beginn der Step- bzw. Start-Funktion mit den entsprechenden Werten geladen.
Die Tastenabfrage realisiert die Tastenentprellung, die Berechnung der Tastenwertigkeit sowie die Ausführung der eingegebenen Funktion.

3.2.2. Funktionen der Datenbehandlung

- Decrement and display memory (Taste DDM)
  Diese Funktion erniedrigt den Programmzählerstand PC der Anwender CPU um 1 und bringt den neuen Programmzählerstand sowie den Inhalt der durch ihn adressierten Speicherzelle über die vier Adreß-LED's bzw. Daten-LED's zur Anzeige.

- Increment and display memory (Taste IDM)
  Diese Funktion erhöht den Programmzählerstand PC der Anwender CPU um 1 und bringt den neuen Programmzählerstand und den Inhalt der durch ihn adressierten Speicherzelle zur Anzeige.

- Set (Taste SET)
  Mittels dieser Funktion kann ein beliebiges CPU-Register der Anwender-CPU oder die durch den Programmzähler adressierten Speicherzelle auf einen neuen Wert gesetzt werden. Der neue Inhalt wird zur Anzeige gebracht.

- Display (Taste DISP)
  Diese Funktion bringt den Inhalt eines beliebigen CPU-Registers oder der durch den Programmzählerstand adressierten Speicherzelle zur Anzeige.

- Fill memory (Taste FILL)
  Diese Funktion ermöglicht das automatische Füllen eines beliebigen zusammenhängenden Speicherbereiches (nur RAM-Speicher) mit einem konstanten Wert. Die Anfangsadresse, Endadresse und der Wert sind anzugeben.

- Input mode (Taste INP)
  Diese Funktion gestattet die forlaufende Eingabe von Werten (Bytes) in den RAM-Speicher, wobei der Programmzählerstand der Anwender-CPU automatisch zur Adressierung der Speicherzellen erhöht wird. Der aktuelle Programmzählerstand und der Inhalt der durch ihn adressierten Speicherzelle werden angezeigt.

3.2.3. Funktionen für die Bedienungsorganisation, Daten- und Adresseneingabe und Register- und Speicheradressierung

- Execution (Taste EX)
  Diese Funktion schließt die Eingabe der für bestimmte Funktionsausführungen notwendige Eingabeketten ab, und bringt die gewünschte Funktion zur Ausführung. Die Funktionstasten SET, DISP, FILL, INP, BRK, STORE, LOAD, MOVE und HEX benötigen zur Ausführung ihrer Funktion den Aufruf der Execution-Funktion.
  Die Funktionstasten DDM, IDM, START, STEP und STORN führen die Execution-Funktion automatisch mit aus.

- Storn (Taste STORN)
  Die Storn-Funktion löscht alle bisherigen Eingaben sowie den Error-LED-Speicher. Wurde bei der Eingabekettenbear-

beitung oder bei der Funktionsausführung ein Fehler erkannt und signalisiert (Error LED leuchtet), so sind nur die Tasten STORN und RESET (Systeminitialisierung) wirksam.

- Reset (Taste RESET)
  Diese Taste bewirkt ein System-Reset, d. h. alle peripheren Schaltkreise mit Reset-Eingang werden rückgesetzt und das Betriebsprogramm wird neu gestartet.

- Hexadezimaltasten (Tasten 0...9, A...F)
  Diese Tasten bewirken in ihrer ersten Funktion die Eingabe von Hexadezimalzahlen.

- Registeradressierung
  Die Tasten I, PC, SP, IX, IY, H, L, A...F adressieren in ihrer zweiten Funktion alle direkt adressierbaren CPU-Register.

- Zusatzregisteradressierung (Taste ')
  Diese Taste adressiert direkt im Zusammenhang mit einer Registeradressierung den Alternativregistersatz.

- Speicheradressierung (Taste M)
  Diese Taste adressiert in ihrer ersten Funktion die vom Programmzähler adressierte Speicherzelle.

- Speicherblockverschiebung (Taste MOVE)
  Diese Funktion ermöglicht das Verschieben von Datenblöcken. Als zusätzliche Information muß die Quell- und Zieladresse sowie die Blocklänge angegeben werden.

- Hexadezimaladdition und -subtraktion (Taste HEX)
  Diese Funktion realisiert die Hexadezimaladdition bzw. -subtraktion zweier 16-bit-Zahlen. Nach der Funktionswahl erfolgt die Eingabe der beiden Zahlen (durch EX abgeschlos-

sen) und die Festlegung +/-. Der anschließende EX-Befehl löst die gewünschte Operation aus. Das Ergebnis erscheint auf dem 4-stelligen Anzeigefeld.

3.2.4. Funktionen für den Anwenderprogrammlauf und -testung

- Step-Funktion (Taste STEP)
  Diese Funktion ermöglicht das befehlsweise Abarbeiten von Anwender-Programmen. Für die Befehlsabarbeitung werden die entsprechenden Werte der Anwender-CPU verwendet und danach entsprechend dem Befehl aktualisiert. Über die Anzeige wird die Adresse des nächsten Befehls ausgegeben.

- Start-Funktion (Taste START)
  Diese Funktion gestattet den automatischen Start eines Anwenderprogramms. Für den Programmstart werden alle Werte der Anwender-CPU (außer I-Register) verwendet. Das Monitor-Betriebsprogramm ist außer der Anzeigebedienung nach erfolgtem Übergang abgeschaltet.

- Breakpoint-Funktion (Taste BRK)
  Mit dieser Funktion erfolgt ein programmierter Rücksprung aus einem Anwenderprogramm in das Monitor-Betriebsprogramm, wobei die Register der virtuellen Anwender-CPU aktualisiert werden. Über die Anzeige wird die Unterbrechungsadresse ausgegeben.

3.2.5. Funktionen für das Kassetten-Interface

- Store on tape (Taste STORE)
  Diese Funktion gestattet das automatische Übertragen von beliebigen Speicherbereichen über eine serielle Schnittstelle auf ein Kassetten-Interface mit einer Übertragungsrate von 110 Baud (110 bit pro Sekunde). Dafür sind die Anfangs- und Endadresse und wahlweise die Startadresse mit

anzugeben. In der Anzeige wird die Adresse des momentan übertragenen Speicherplatzes ausgegeben.

- Load from tape (Taste LOAD)
  Die Funktion ermöglicht die Übertragung von Daten über eine serielle Schnittstelle von einem Kassetten-Interface in den RAM-Speicher. In der Anzeige wird die Adresse des Speicherplatzes des momentan empfangenen Datenbytes ausgegeben. Erfolgt der Befehl LOAD im Zusammenhang mit GO, wird das Programm automatisch ab der empfangenen Startadresse gestartet. Die Funktion LOAD im Zusammenhang mit '?' führt ein Prüflesen der Kassette ohne Speicherveränderung durch.

3.3. Hinweise zu einigen Funktionen

Die Step- und Start-Funktionen bedingen in ihrer Ausführung das Setzen des Programmzählers PC und des Stackpointers SP. Der Programmzähler wird auf die Startadresse des Anwenderprogramms geladen. Der Stackpointer erhält eine Adresse im frei verfügbaren RAM-Speicherbereich mit einem ausreichenden Abstand vom Programm. Wird der Stackpointer auf den Beginn des Monitorstacks (0F3B0H) gelegt, so wird mit der Rückkehr in den Monitor der Anwenderstack überschrieben und eine Kontrolle der Stackbelegung des Anwenderprogrammablaufes nicht mehr möglich.
Die Rückkehr in den Monitor-Mode ist bei automatischen Anwenderprogrammlauf auf folgende Arten möglich:

1. programmierter Sprung in den Monitor
2. vorgewählte Unterbrechung (Breakpoint-Funktion)
3. Abbruch durch Reset-Funktion

Der Sprung in den Monitor (CALL 001EH) bewirkt den normalen Ablauf des Monitorprogrammes im Anschluß an das Anwenderprogramm. Die Anzeige wird dadurch nicht beeinflußt.
Die Breakpoint-Funktion erlaubt die Rückkehr in den Monitor

bei einer vorwählbaren Adresse des Anwenderprogrammes, unter der Voraussetzung, daß sich das Anwenderprogramm im RAM-Speicher befindet. Der Breakpoint muß auf das erste Byte eines Befehls gesetzt werden. Beim Setzen des Breakpointes wird das erste Byte des Befehls gegen einen Restart 18-Befehl ausgetauscht. Der Rücktausch erfolgt mit Start bzw. Step im Anschluß an die Unterbrechung ohne Adresseneingabe sowie beim Löschen des Breakpoints ("leere" Breakpoint-Funktion). In eine noch nicht gelöschte Unterbrechung darf kein Reset erfolgen, da kein Rücktausch des Originalbefehlskodes erfolgt und die Breakpointdaten (Adresse und Befehlskode) durch die Initialisierung verlorengehen.
Die Store-Funktion überträgt außer den Daten des vorgegebenen Speicherbereiches die Start- und Endadresse sowie wahlweise die GO-Adresse. Beim Empfang der Daten mittels der LOAD-Funktion werden die Start- und Endadresse gemäß dem geladenen Programm übernommen. Die LOAD/GO-Funktion erlaubt den Start des Programmes ab der empfangenen Startadresse unmittelbar im Anschluß an den Empfang. Ein nachträglicher Vergleich des Programms ist mittels der LOAD?-Funktion möglich.
Das Verschieben von Blöcken im Speicher ermöglicht die MOVE-Funktion, wobei die Quell- und Zieladresse sowie die Blocklänge anzugeben ist.

3.4. Monitorbedienungsanweisung

Das Lernsystem-Monitor-Programm arbeitet auf der Basis von Eingabeketten. Den Abschluß einer Eingabe bildet eine Taste mit sofortiger Wirkung; dies sind die Funktionstasten EX, STORN, STEP, START, IDM und DDM. Werden Ketten durch andere Funktionstasten eröffnet, benötigen diese zur Funktionsausführung die Bestätigung der EX-Taste.
Die Länge einer Eingabekette ist auf 12 Eingaben begrenzt. Wird diese Länge überschritten, erfolgt eine Fehlersignalisierung mittels Error-LED. Adreß- und Dateneingaben werden

auf 4 bzw. 2 Eingaben begrenzt. Der Monitor erkennt im Überlauffall die jeweils erste Adreß- oder Dateneingabe als Tastenfehlbedienung und verarbeitet die 2. bis 5. bzw. 2. und 3. Werteingabe. Bei verkürzten Eingaben werden die jeweiligen Adressen bzw. Daten automatisch durch Führungsnullen ergänzt.

Die Tasten STORN, START, STEP, IDM, DDM bewirken die sofortige Ausführung der entsprechenden Funktion. Die EX-Taste ist nur in Verbindung mit weiteren Funktionstasten (außer Abschluß des Input-Modes) sinnvoll und bringt die vorher eingegebene Funktion zur Ausführung.

Beispiel:

- Setzen des Programmzählers PC:
  SET PC Hz Hz Hz Hz EX (Hz = Hexadezimalzahl)

- Setzen des durch PC adressierten Speicherplatzes:
  SET M HZ HZ EX

- Setzen des Registers A
  SET A HZ HZ EX

- Anzeige des Stackpointers SP:
  DISP SP EX

- Füllen des Speicherbereiches F000H FOFFH mit der Konstanten 47H:
  FILL F 000 EX F 0 F F EX 4 7 EX EX

- Eingabe mittels Input-Mode:
  INP Hz Hz EX Start in den Input-Mode und erste Eingabe
  Hz Hz EX zweite Eingabe
  Hz Hz EX EX dritte Eingabe und Abschluß des Input-Mode

- Setzen eines Breakpoints:
  BRK Hz Hz Hz Hz EX

- Löschen eines Breakpoints:
  BRK EX

- Laden des Speicherbereiches F000H ... F100H auf Kassette:
  STORE F 0 0 0 EX F 1 0 0 EX EX

- Laden eines Programmes mit automatischem Start ab der empfangenen Startadresse:
  LOAD GO EX

- Hexadezimaladdition:
  HEX 7A31 EX 1B70 EX + EX

3.5. Universelle Unterprogramme

Das Lernsystem-Monitorprogramm enthält mehrere Unterprogramme, welche ebenfalls vom Anwender nutzbar sind.

TABFR: Tastaturabfrage
Abfrage der Tastatur bis eine korrekte Tastenbetätigung erkannt wird.
Eintritt: Kein notwendiger Vorbereitungswert

Austritt: Tastenwertigkeit in Register A
Startadresse: 0155H

FFSET: Fehlerbit setzen
Austritt: Bit 7 von F3F9H gesetzt
Startadresse: 021DH

FFCLR: Fehlerbit löschen
Austritt: Bit 7 von F3F9H gelöscht
Startadresse: 0225H

DISON: Einschalten des Anzeige-Interrupts
Startadresse: 0138H

DISOFF: Abschalten des Anzeige-Interrupts
Startadresse: 014OH

MSEK: Zeitschleife
Eintritt: BC enthält Faktor (BC x 8,4 µs)

DISDAT: Anzeige eines Datenbytes
Eintritt: Datenbyte in A
Startadresse: 01DC

DISADR: Anzeige einer Adresse
Eintritt: Adresse in HL
Startadresse: 01E8H

SEGUM:
Eintritt: Hex-Zahl in A, (A0...A3 beliebig, A4...A7 = 0)
Rückkehr: 7-Segment-Kode in A
Startadresse: 0214H

3.6. Programmierhinweise

Alle peripheren Schaltkreise werden durch das Lernsystem-Monitorprogramm nicht initialisiert (außer bei Aufruf der STORE bzw. LOAD-Funktion).
Wird in einem Anwenderprogramm ein HALT-Befehl verwendet, so erfolgt kein ständiger Halt der CPU, da der Anzeige-Interrupt (NMI) die CPU aus dem HALT-Zustand bringt. In diesem Fall muß eine Halt-Schleife verwendet werden:

```
Op-code      Mnemonik
76           M1: HALT
18 FD        JR M1
```

Soll ein Anwenderprogramm im Echtzeitbetrieb ablaufen, so muß der Anzeige-Interrupt abgeschaltet werden (CALL DISOFF), da dieser NMI den normalen Programmablauf regelmäßig für eine nicht unwesentliche Dauer unterbricht.

4. Befehlsbeschreibung des U 880

4.1. Befehlsstruktur

Ein Befehl ist eine Anweisung an den Rechner, eine bestimmte Operation auszuführen. Er besteht aus Operationsteil (1 -3 Byte) und Adreßteil (1 - 2 Byte). Dabei sind folgende Befehlsstrukturen möglich:

1-Byte-Befehl

Op-Code

Bsp.: 04 — Erhöhe Inhalt von Register B

2-Byte-Befehl

Op-Code	Op-Code

Bsp.: CB 40 — Ist Bit 0 von Reg.B gesetzt?

Op-Code	Direktoperand

Bsp.: 3E 05 — Lade Register A mit 05

3-Byte-Befehl

Op-Code	Op-Code	Direktoperand

Bsp.: DD 77 3 — Lade Register A mit Inhalt der Adresse IX + 3

Op-Code	Adresse	Adresse

Bsp.: 21 00 50 — Lade Doppelregister HL mit Adresse 5000H

4-Byte-Befehl

Op-Code	Op-Code	Adresse	Adresse

Bsp.: DD 21 00 50 — Lade Indexregister IX mit Adresse 5000H

	Op-Code	Op-Code	Direktoperand	Direktoperand
Bsp.:	DD	36	03	05

Lade Inhalt von Register IX + 3 mit 05

	Op-Code	Op-Code	Direktoperand	Op-Code
Bsp.:	DD	CB	03	46

Ist Bit 0 von Inhalt der Adresse IX + 3 gesetzt

Für jeden Befehl existiert ein mnemonischer Operationscode (Kürzel). Damit wird die Programmierung für den Anwender einfacher und effektiver als in maschineninterner Form. Der hier verwendete mnemonische Code entspricht der Assemblersprache SYPS K 1520.

Beispiele:

Maschinencode	Mnemonik
04	INC B
CB 40	Bit 0, B
3E 05	LD A, 5
DD 77 3	LD (IX+3), A
21 00 50	LD HL, 5000H
DD 21 00 50	LD IX, 5000H
DD 36 03 05	LD (IX + 3), 5
DD CB 03 46	Bit 0, (IX + 3)

Übung:

Suchen Sie in der Befehlskarte zu folgenden Ausdrücken

a) den Maschinencode

INC HL

LD DE, 8976H

Bit 3, (IX + 7)

b) die Mnemonik

```
3D
31  47  39
DD  CB  07  FE
```

4.2. Adressierungsarten

Mit Hilfe der Befehle, werden Daten verarbeitet, die auf CPU-Registern bzw. externen Speichern abgespeichert sind, auf E/A-Kanälen anliegen oder durch den Programmierer direkt im Befehl angegeben werden. Ein Zugriff auf diese Daten erfolgt durch die entsprechende Adressierung. Im Befehlssatz kommen folgende 6 Adressierungsarten zum Einsatz:

- 1-Byte-Direktoperand
  Der Operand wird direkt im Befehl angegeben.
  Bsp.: LD A, 5

- 2-Byte-Direktoperand
  Die Adresse wird direkt im Befehl angegeben. Die Abspeicherung erfolgt in der Reihenfolge niederwertiger Teil vor höherwertigen Teil der Adresse.
  Bsp.: LD HL, 5000H
  ⋮
  Maschinencode
  21 00 50

- Speicheradressierung über Register
  Die Adresse steht in einem vorher eingestellten Registerpaar.
  Bsp.: LD HL, 5000H
  LD A, (HL)
  ⋮

- Registeradressierung

  Der Befehl bezieht sich auf ein Register bzw. Registerpaar.

  Bsp.:
```
LD  C, 4
LD  A, C
 :
 :
```
  oder
```
LD  HL, 5000H
LD  SP, HL
 :
 :
```

- Direkte Speicheradressierung

  Die Speicheradresse wird im Befehl als Direktwert angegeben.

  Bsp.:
```
LD  A,  (5000H)
 :
 :
```

- Indizierte Adressierung

  Mit dieser Adressierungsart wird dem Programmierer die Möglichkeit gegeben, eine Modifizierung der Adresse (Verschiebung) um + 127 bzw. - 128 Byte vorzunehmen.

  Bsp.:
```
LD  HL,  5003H
LD  (HL),   4
LD  IX,  5000H
LD  A,  (IX + 3)
 :
 :
```

- Relative Adressierung

  Bei Befehlen mit relativer Adressangabe wird im Adreßteil nur die sogenannte Distanzadresse (1 Byte) angegeben. Sie kennzeichnet den Abstand zum aktuellen Befehlszählerstand. Der Abstand kann zwischen + 127 und - 128 Byte groß sein. Bei Verwendung der relativen Adressierung in Sprungbefehlen kann jeweils ein Byte des Adreßteils eingespart werden.

  Bsp.:

  a) Direktoperand
```
PC = 5000H
5000H: JPM 5004H
        :
        :
```
  b) relative Adressierung
```
PC = 5000H
5000H: JR + 4
        :
        :
```

4.3. Flag-Bit-Technik

Die meisten Befehle des U 880 beeinflussen das F-Register, d. h. sie verändern entsprechend dem Operationsergebnis bestimmte Flags.
Diese Flags können über bedingte Sprungbefehle bzw. Unterprogrammaufrufe oder -rücksprünge zu Programmverzweigungen ausgewertet werden. Das sogenannte Flagregister hat dabei folgenden Aufbau

	D7	D6	D5	D4	D3	D2	D1	D0
Flagregister	S	Z	X	H	X	P/V	N	CY

S - Vorzeichenbit (Sign.Flag)
Z - Nullbit (Zero-Flag)
H - Halbbyteüberlaufbit (Half-Carry-Flag)
PV - Paritäts/Überlauf-Bit (Parity/Overflow-Flag)
N - Additions/Subtraktions-Bit (Add/Subtract-Flag)
CY - Übertragsbit (Carry-Flag)
X - nicht belegt

Im folgenden werden die einzelnen Flags kurz beschrieben. Die Stellung der Flags bei den einzelnen Befehlen ist der Befehlskarte zu entnehmen.

1. Vorzeichenbit (S-Flag)
Das Vorzeichenbit wertet bei bestimmten Befehlen stets das höchstwertigste Bit des jeweiligen Registers aus. Nach Arithmetikbefehlen kennzeichnet ein gesetztes S-Flag (S = 1) eine negative Zahl und S = 0 eine positive Zahl. Die restlichen sieben Bit stellen bei einer positiven Zahl den Betrag dar. Eine negative Zahl ergibt sich hingegen aus der Zweierkomplement-Darstellung der zugehörigen positiven Zahl. Dies wird durch die Negation und nachfolgender Addition mit eins realisiert.

Bsp.:

	+ 5		- 5	
	0000 0101		1111 1010	Negation
			0000 0001	+ 1
			1111 1011	

Übung: Stellen Sie die Zahlen + 118 und - 118 binär dar!

2. Nullbit (Z-Flag)

Das Z-Flag wird in folgenden Fällen gesetzt:

- nach arithmetischen und logischen Operationen, deren Ergebnis Null ist
- bei Bitbefehlen zur Kennzeichnung eines gesetzten oder nicht gesetzten Bits
- bei Vergleichs- und Suchbefehlen mit positivem Ergebnis
- bei IN/OUT-Befehlen, falls einerseits bei Datenübertragungen das Zählregister 0 wird bzw. andererseits die am Eingangstor anliegenden Daten den Wert 0 haben.

3. Halbbyte-Übertragsbit (H-Flag)

Das H-Flag wird nach arithmetischen Operationen gesetzt, sobald ein Überlauf von Bit D3 auf Bit D4 auftritt. Das wird beim Befehl DAA genutzt, um das Ergebnis einer Addition bzw. Subtraktion von zwei gepackten BCD-Zahlen zu korrigieren. Bei der Addition wird das H-Flag beim Übertrag von Bit 3 zu Bit 4 und bei der Subtraktion beim negativen Übertrag von Bit 4 auf Bit 3 gesetzt.

4. Paritäts-/Überlaufbit (P/V-Flag)

Das P/V-Flag wird in folgenden Fällen gesetzt:

- bei logischen Operationen und Verschiebebefehlen, falls Parität am Ergebnisbyte vorliegt (z. B. Anzahl der gesetzten Bits 0, 2, 4, 6, 8)

- bei arithmetischen Befehlen, falls im Resultat der zulässige Zahlenbereich, d. h. größer + 127 bzw. kleiner - 128, überschritten wird

- bei Transport- u. Suchbefehlen, falls der Bytezähler ungleich Null ist.

Bsp.:	0110	0001	97
	0010	0001	33
	1000	0010	130

5. Additions-/Subtraktionsbit (N-Flag)

Das N-Flag wird vom DAA-Befehl ausgewertet. Bei allen Additionsbefehlen wird es rückgesetzt und bei den Subtraktionsbefehlen gesetzt.

6. Übertragsbit (CY-Flag)

Das Übertragungsflag wird bei Additionsbefehlen mit Übertrag bzw. bei Subtraktionsbefehlen mit negativen Übertrag gesetzt. Dadurch ist eine Addition bzw. Subtraktion über mehrere Bytes möglich.

Bsp.: Addition

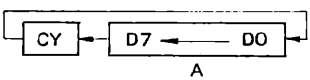
2. Byte		1. Byte		
0000 0001		0011 1000		312
0000 0010		1110 0111		743
0000 0100	CY = 1	0001 1111		1055

Bsp.: Subtraktion

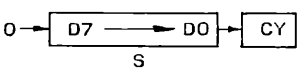
0000 0100	0001 1111	1055	
0000 0010	1110 0111	743	
1111 1101	0001 1001	Zweierkomplement	
0000 0100	0001 1111		
1111 1101	0001 1001		
0000 0001	0011 1000	312	
CY = 1			

Bei den Verschiebebefehlen wird das CY-Flag als Zwischenspeicher für die Übertragung des niederwertigen bzw. höherwertigen Bits eines Datenbytes genutzt.

Bsp.: RLA



SRL



Desweiteren beeinflußt der CMP-Befehl das CY-Flag folgendermaßen:

CMP r (r = CPU-Register)

Cy-Flag	
0	A = 0
0	A > r
1	A < r

Die logischen Befehle AND, OR und XOR setzen das CY-Flag zurück. Der Befehl SCF setzt das CY-Flag und mit CCF kann das Komplement gebildet werden.

4.4. Befehlssatz des U 880

4.4.1. 8-Bit-Ladebefehle

LD r_1, r_2 Laden Register r_1 mit Register r_2
r_1: A, B, C, D, E, H, L
r_2: A, B, C, D, E, H, L

LD r, (HL) Laden Register r mit Inhalt des durch Registerpaar HL adressierten Speicherplatzes
r: A, B, C, D, E, H, L

LD r, n Laden Register r mit einem Direktoperanden
Bsp.: LD C, 14 H
r: A, B, C, D, E, H, L

LD r,(IX+d) Laden Register r mit Inhalt des durch IX plus Verschiebung d adressierten Speicherplatzes
Bsp.: LD A, (IX+3)
r: A, B, C, D, E, H, L

LD r,(IY+d) Laden Register r mit Inhalt des durch IY plus Verschiebung d adressierten Speicherplatzes

	Bsp.: LD H, (IY+11) r: A, B, C, D, E, H, L
LD (HL), r	Laden des durch HL adressierten Speicherplatzes mit Datenbyte aus Register r Bsp.: LD (HL), B bzw. LD M, B r: A, B, C, D, E, H, L
LD (HL),	Laden des durch HL adressierten Speicherplatzes mit einem Direktoperanden Bsp.: LD (HL), 2
LD (IX + d), r	Laden des durch IX plus Verschiebung d adressierten Speicherplatzes mit Datenbyte eines Registers r Bsp.: LD (IX + 3), A r: A, B, C, D, E, H, L
LD (IY + d), r	Laden des durch IY plus Verschiebung d adressierten Speicherplatzes mit Datenbyte aus Register r r: A, B, C, D, E, H, L
LD (IX + d),	Laden des durch IX plus Verschiebung d adressierten Speicherplatzes mit einem Direktoperanden n
LD (IY + d),	Laden des durch IY plus Verschiebung d adressierten Speicherplatzes mit einem Direktoperanden n
LD A, (BC)	Laden des Akkumulators mit Inhalt des durch BC adressierten Speicherplatzes
LD A, (DE)	Laden des Akkumulators mit Inhalt des durch DE adressierten Speicherplatzes

LD A, (nn)	Laden des Akkumulators mit Inhalt des durch nn adressierten Speicherplatzes Bsp.: LD A, (4000H) nn: 16-Bit-Konstante
LD (BC), A	Laden des durch BC adressierten Speicherplatzes mit Datenbyte aus Akkumulator
LD (DE), A	Laden des durch DE adressierten Speicherplatzes mit Datenbyte aus Akkumulator
LD I, A	Laden des Interruptvektor-Registers mit Datenbyte aus Akkumulator
LD R, A	Laden des Refresh-Registers mit Datenbyte aus Akkumulator
LD A, I	Laden des Akkumulators mit Datenbyte aus Interrupt-Vektor-Register
LD A, R	Laden des Akkumulators mit Datenbyte aus Refresh-Register

Übung:

1. Suchen Sie zu den folgenden Mnemonika den jeweils entsprechenden Maschinencode!

```
LD    B, C
LD    B, (HL)
LD    C, 14H
LD    D, (IX + 3)
LD    H, (IY + 11)
LD    (HL), B
LD    (HL), 12
LD    (IX-2), B
LD    (IY+6), D
LD    (IX+1), OFFH
LD    (IY+12), 1
LD    A, (BC)
LD    A, (DE)
LD    A, (4000H)
```

LD	(BC), A
LD	(DE), A
LD	I, A
LD	A, I

4.4.2. 16-Bit-Ladebefehle

LD dd, nn	Laden des Doppelregisters dd mit 16-Bit-Konstante dd: BC, DE, HL, SP
LD IX, nn	Laden des Registers IX mit 16-Bit-Konstante
LD IY, nn	Laden des Registers IY mit 16-Bit-Konstante
LD HL, (nn)	Laden des Registerpaares HL mit Inhalt der durch nn und nn+1 adressierten Speicherplätze (nn+1) → H (nn) → L
LD dd, (nn)	Laden des Registerpaares dd mit Inhalt der durch nn und nn+1 adressierten Speicherplätze (nn+1) → d_H (höherwertiger Teil) (nn) → d_L (niederwertiger Teil) dd: BC, DE, SP
LD IX, (nn)	Laden des Registers IX mit Inhalt der durch nn und nn+1 adressierten Speicherplätze (nn+1) → IX_H (nn) → IX_L
LD IY, (nn)	Laden des Registers IY mit Inhalt der durch nn und nn+1 adressierten Speicherplätze (nn+1) → IY_H (nn) → IY_L

LD (nn), HL — Laden des durch nn adressierten Speicherplatzes mit Doppelregister HL

$H \longrightarrow (nn+1)$

$L \longrightarrow (nn)$

LD (nn), dd — Laden des durch nn adressierten Speicherplatzes mit Doppelregister dd

$dd_H \longrightarrow (nn+1)$

$dd_L \longrightarrow (nn)$

dd: BC, DE, SP

LD (nn), IX — Laden des durch nn adressierten Speicherplatzes mit Register IX

$IX_H \longrightarrow (nn+1)$

$IX_L \longrightarrow (nn)$

LD (nn), IY — Laden des durch nn adressierten Speicherplatzes mit Register IY

$IY_H \longrightarrow (nn+1)$

$IY_L \longrightarrow (nn)$

LD SP, HL — Laden des Stackpointers mit Doppelregister HL

$H \longrightarrow SP_H$

$L \longrightarrow SP_L$

LD SP, IX — Laden des Stackpointers mit Register IX

$IX_H \longrightarrow SP_H$

$IX_L \longrightarrow SP_L$

LD SP, IY — Laden des Stackpointers mit Register IY

$IY_H \longrightarrow SP_H$

$IY_L \longrightarrow SP_L$

Beispiele:

1. Laden des Speicherplatzes F000H mit dem Wert 30H

```
LD          HL, ØF000H
LD          A, 30H
LD          (HL), A
            :
```

2. Laden des Speicherplatzes F000H mit 30H und des Speicherplatzes F001H mit 31H.

```
a) LD       HL, 3130H
   LD       (ØF000H), HL
            :

b) LD       IX, F000H
   LD       A, 30H
   LD       (IX), A
   LD       A, 31H
   LD       (IX + 1), A
            :
```

Übung:

Laden Sie über das Doppelregister BC den Inhalt des Speicherplatzes F100H mit 12 und bringen Sie diesen Wert in das Doppelregister HL (Mnemonik und Maschinencode).

4.4.3. Registertauschbefehle

EX DE, HL — Austausch der Register DE, HL

D ⟷ H

E ⟷ L

EXAF — Austausch von Registersatz AF mit AF'

A ⟷ A'

F ⟷ F'

EXX — Austausch der Registerpaare BC mit BC', DE mit DE' und HL mit HL'

BC ↔ BC'
DE ↔ DE'
HL ↔ HL'

EX (SP), HL — Austausch des Registers HL mit Inhalt des Stackpointers SP

H ↔ (SP+1)
L ↔ (SP)

EX (SP), IX — Austausch des Registers IX mit Inhalt des Stackpointers SP

IX_H ↔ (SP+1)
IX_L ↔ (SP)

EX (SP), IY — Austausch des Registers IY mit Inhalt des Stackpointers SP

IY_H ↔ (SP+1)
IY_L ↔ (SP)

Beispiel: Verändern der im Stackpointer abgespeicherten Rückkehradresse (neue Adresse steht im Register DE)

```
LD      DE, 0F000H
EX      DE, HL
EX      (SP), HL
 :
```

Übung: Erstellen Sie ein Programm, welches die Strichregister mit folgenden Daten lädt (Mnemonik, Maschinencode):

A'	-	05
B'	-	31
C'	-	17
D'	-	33
E'	-	0FAH
H'	-	17H
L'	-	13

4.4.4. 8-Bit-Arithmetikbefehle

Die 8-Bit-Arithmetikbefehle beeinflussen im Ergebnis der Operation die entsprechenden Flags. Die Operationen finden stets zwischen dem Akkumulator und einem zweiten Datenbyte statt, welches sowohl als Direktoperand angegeben werden kann bzw. in einem Register steht, oder durch dieses adressiert wird. Das Ergebnis steht immer im Akkumulator. Der zweite Operand bleibt unverändert erhalten. Eine Ausnahme bilden die Inkrement- und Dekrementbefehle. Sie erhöhen oder erniedrigen den Operanden um 1. Der Operand kann entweder ein Register oder der Inhalt einer indirekt oder indiziert adressierten Speicherzelle sein.

ADD r	Addition des Registerinhaltes r zum Akkumulatorinhalt r: A, B, C, D, E, H, L
ADD M	Addition des durch Registerpaar HL adressierten Speicherplatzinhaltes zum Akkumulatorinhalt
ADD n	Addition eines Direktoperanden zum Akkumulatorinhalt
ADD (IX+d)	Addition des durch Register IX plus Verschiebung d adressierten Speicherplatzinhaltes zum Akkumulatorinhalt (- 128 = d 127)
ADD (IY+d)	Addition des durch Register IY plus Verschiebung d adressierten Speicherplatzinhaltes zum Akkumulatorinhalt (- 128 = d = 127)
ADC r	Addition des Registerinhaltes plus Carry-Flag zum Akkumulatorinhalt

ADC M	Addition des durch Doppelregister HL adressierte Speicherplatzinhaltes plus Carry-Flag zum Akkumulatorinhalt r: A, B, C, D, E, F, H, L
ADC n	Addition eines Direktoperanden plus Carry-Flag zum Akkumulatorinhalt
ADC (IX+d)	Addition des durch Register IX plus Verschiebung d adressierten Speicherplatzinhaltes zum Akkumulatorinhalt (- 128 = d = 127)
ADC (IY+d)	Addition des durch Register IY plus Verschiebung d adressierten Speicherplatzinhaltes zum Akkumulatorinhalt (- 128 = d = 127)
SUB r	Subtraktion des Registerinhaltes r vom Akkumulatorinhalt r: A, B, C, D, E, H, L
SUB M	Subtraktion des durch das Doppelregister HL adressierten Speicherplatzinhaltes vom Akkumulatorinhalt
SUb n	Subtraktion eines Direktoperanden vom Akkumulatorinhalt
SUB (IX + d)	Subtraktion eines durch Register IX plus Verschiebung d adressierten Speicherplatzinhaltes vom Akkumulatorinhalt (- 128 = d = 127)
SUB (IY + d)	Subtraktion eines durch Register IY plus Verschiebung d adressierten Speicherplatzinhaltes vom Akkumulator (- 128 = d = 127)

SBC r	Subtraktion des Registerinhaltes r plus Carry-Flag vom Akkumulatorinhalt r: A, B, C, D, E, H, L
SBC M	Subtraktion des durch Doppelregister HL adressierten Speicherplatzinhaltes plus Carry-Flag vom Akkumulatorinhalt
SBC n	Subtraktion eines Direktoperanden n plus Carry-Flag vom Akkumulatorinhalt
SBC (IX + d)	Subtraktion des durch das Register IX adressierten Speicherplatzinhaltes plus Carry-Flag vom Akkumulatorinhalt (- 128 = d = 127)
SBC (IY + d)	Subtraktion des durch das Register IY adressierten Speicherplatzinhaltes plus Carry-Flag vom Akkumulatorinhalt (- 128 = d - 127)
INC r	Erhöhung des Registerinhaltes um 1 r: A, B, C, D, E, L, H
INC M	Erhöhung des durch Doppelregister HL adressierten Speicherplatzes um 1
INC (IX + d)	Erhöhung des durch Register IX plus Verschiebung d adressierten Speicherplatzinhaltes um 1 (- 128 d - 127)
INC (IY + d)	Erhöhung des durch Register IY plus Verschiebung d adressierten Speicherplatzinhaltes um 1 (- 128 d - 127)

DEC	Verminderung des Registerinhaltes r um 1 r: D, B, C, D, E, H, L
DEC M	Verminderung des durch Doppelregister HL adressierten Speicherplatzinhaltes um 1
DEC (IX + d)	Verminderung des durch Register IX plus Verschiebung d adressierten Speicherplatzinhaltes um 1 (- 128 = d = 127)
DEC (IY + d)	Verminderung des durch Register IY plus Verschiebung d adressierten Speicherplatzinhaltes um 1 (- 128 = d = 127)

<u>Beispiele:</u>

a) Addition zweier Zahlen (Summe kleiner 256)

- 1. Zahl steht auf Adresse F000H
- 2. Zahl steht auf Adresse F100H
- 3. Ergebnis soll wieder auf Adresse F000H abgespeichert werden.

```
LD      DE, ØFØØØH
LD      A,  (DE)
LD      HL, ØF100H
ADD     M
LD      (DE), A
        :
        :
```

b) Subtraktion zweier 2-Byte Zahlen mit Hilfe von 8-Bit-Arithmetikbefehlen u. Verwendung d. Indexregister

- Subtrahend beginnt auf Adresse FOOOH (2 Byte)
- Minuend beginnt auf Adresse F1OOH (2 Byte)
- Ergebnis soll wieder ab Adresse FOOOH gespeichert werden.

```
LD      IX, ØFOOOH
LD      IY, ØF1ØØH
LD      A,  (IX)
SUB     (IY)
LD      (IX), A
LD      A,  (IX + 1)
SBC     (IY + 1)
LD      (IX + 1), A
        :
```

c) Erhöhen Sie den Inhalt des Speicherplatzes FOOOH von 2 auf 4

```
LD      HL, ØFOOOH
LD      M, 2
INC     M
INC     M
        :
```

<u>Übung:</u>

1. Überprüfen Sie die Beispiele a), b) und c) mit folgenden Werten auf dem Lernsystem

   a) 1. Zahl: 6
      2. Zahl: 27

   b) Subtrahend: 1055
      Minuend 734

   c) Inhalt (HL) soll von 10 auf 12 erhöht werden.

2. Schreiben Sie ein Programm für folgende Operationen "312 + 743" mit den bisher erläuterten Befehlen und speichern Sie das Ergebnis auf Adresse F200H ab.

4.4.5. 16-Bit-Arithmetikbefehle

ADD HL, dd	Addition von Registerpaar dd zu Registerpaar HL dd: BC, DE, HL, SP
ADD IX, pp	Addition von Registerpaar pp zu Register IX pp: IX, BC, DE, SP
ADD IY, pp	Addition von Registerpaar pp zu Register IX pp: IY, BC, DE, SP
ABC HL, dd	Addition von Registerpaar pp plus Carry-Flag zu Registerpaar HL dd: BC, DE, HL, SP
SBC HL, dd	Subtraktion des Doppelregisters dd plus Carry-Flag vom Registerpaar HL dd: BC, DE, HL, SP
INC dd	Erhöhung des Doppelregisterwertes um 1 dd: BC, DE, HL, SP, IX, IY
DEC dd	Verminderung des Doppelregisterwertes um 1 dd: BC, DE, HL, SP, IX, IY

Bei den INC- und DEC-Befehlen der Doppelregister werden keine Flags gesetzt.

Beispiele:

a) Subtrahieren Sie 734 von 1055 und speichern Sie diesen Wert ab Adresse F000H

```
LD      HL, 1055
LD      DE,  734
SBC     HL, DE
LD      (ØFØØØH), HL
         :
```

b) Laden Sie ab Adresse F100H 3 Byte mit der Zahl 7

```
LD      HL, ØF100H
LD      A, 7
LD      M, A
INC     HL
LD      M, A
INC     HL
LD      M, A
         :
```

Übung:

1. Überprüfen Sie die Beispiele a) und b) mit dem Lernsystem!

2. Entwickeln Sie ein Additionsprogramm für folgende Operation "312 + 743" mit 2 Byte-Arithmetikbefehlen und speichern Sie das Ergebnis auf Adresse F200H ab!

4.4.6. Sprungbefehle

JMP nn unbedingter Sprung nach Adresse nn
        Bsp.: JMP 8100H

JPNZ nn Sprung nach Adresse nn, wenn Z-Flag gleich 0 ist.

JPZ Sprung nach Adresse nn, wenn Z-Flag gleich 1 ist.

JPNC nn	Sprung nach Adresse nn, wenn CY-Flag gleich 0 ist
JPC	Sprung nach Adresse nn, wenn CY-Flag gleich 1 ist
JPPO nn	Sprung nach Adresse nn, wenn P/V-Flag gleich 0 ist
JPPE nn	Sprung nach Adresse nn, wenn P/V-Flag gleich 1 ist
JPP	Sprung nach Adresse nn, wenn S-Flag gleich Ø ist
JPM	Sprung nach Adresse nn, wenn S-Flag gleich 1 ist
JR e	unbedingter relativer Sprung e: Sprungweite zwischen aktuellen Befehlszählerstand und Zieladresse (- 126 = e = 129)
JRNZ e	relativer Sprung mit der Sprungweite e, wenn Z-Flag gleich 0 ist (- 126 = e = 129)
JRZ e	relativer Sprung mit der Sprungweite e, wenn Z-Flag gleich 1 ist (- 126 = e = 129)
JRNC e	relativer Sprung mit der Sprungweite e, wenn CY-Flag gleich 0 ist (-126 = e = 129)
JRC e	relativer Sprung mit der Sprungweite e, wenn CY-Flag gleich 1 ist (- 126 = e 129)
JMP M	unbedingter Sprung zur Adresse, die durch Doppelregister HL festgelegt ist Bsp.: LD HL, 8000H JMP M (Sprung zur Adresse 8000H)

JMP (IX) unbedingter Sprung zur Adresse, die durch Doppelregister IX festgelegt ist

JMP (IY) unbedingter Sprung zur Adresse, die durch Doppelregister IY festgelegt ist

DJNZ e Verminderung des Inhaltes von Register B. Falls dabei der Inhalt des Registers ungleich 0 ist, erfolgt ein Sprung zur Adresse, die sich aus der Sprungweite e ergibt. Andernfalls wird mit dem nächsten Befehl fortgesetzt.

<u>Beispiele:</u> a) Addieren Sie den Inhalt der Speicherplätze ADR1, ADR2 und laden Sie bei Überlauf die Fehlerzelle FEHLZ mit 1. Das Ergebnis der Operation soll auf Adresse ADR3 abgespeichert werden. (ADR1, ADR2 u. ADR3 je 1 Byte)

```
        LD      A, (ADR1)
        LD      HL, ADR2
        ADD     M
        LD      HL, FEHLZ
        JRC     MARK 1
        LD      (ADR3), A
        LD      M, Ø
        JMP     END
MARK 1: LD      M, 1
        JMP     END
```

b) Laden Sie ab Adresse ADR1 die folgenden 10 Speicherplätze mit 1 und setzen Sie nach erfolgter Operation die Speicherzelle HZELL auf 1.

```
            LD      HL, ADR1-1
            LD      A,  1
            LD      B,  10
MARK 2:     INC     HL
            LD      M,  A
            DJNZ    MARK2
            LD      A,  1
            LD      (HZELL), A
                    :
```

Übung:

1. Überprüfen Sie die Beispiele a) und b) mit selbstgewählten Werten und Adressen!
2. Entwickeln Sie ein Programm für eine Warteschleife mit einer Dauer von ca. 1 Sekunde (Taktzeit 0,4 µs)!

4.4.7. 1-Byte-Logik-Befehle

Die im Befehlsvorrat enthaltenen logischen Funktionen lassen sich in folgenden Wahrheitstabellen beschreiben:

AND s
(logisches UND zweier Bits ist nur 1, wenn beide Bits 1 sind)

S	A	O
0	0	0
0	1	0
1	0	0
1	1	1

ORs
(logisches ODER zweier Bits ist nur dann 0, wenn beide Bits 0 sind)

S	A	O
0	0	0
0	1	1
1	0	1
1	1	1

XOR S

(Antivalenz ist dann gleich 1, wenn die Bits ungleich sind)

S	A	Q
0	0	0
0	1	1
1	0	1
1	1	Ø

AND s

Logisches UND von S mit dem Akkumulator

S: - Register A, B, C, D, E, H, L
- Direktoperand n
- Speicherinhalt, der durch die Register HL, IX+d, IY+d festgelegten Adresse

Bsp.: AND B

A: 0001 0111
B: 0010 0101
A: 0000 0101

OR s

Logisches ODER von S mit dem Akkumulator (S wie bei AND S)

Bsp.: OR B

A: 0001 0111
B: 0010 0101
A: 0011 0111

XOR S

Exklusives ODER von S mit dem Akkumulator (S wie bei AND S)

Bsp.: XOR B

A: 0001 0111
B: 0010 0101
A: 0011 0010

CMP S Vergleich von S mit dem Akkumulator. Die Inhalte von A und S bleiben erhalten. Die Flags werden je nach Ergebnis folgendermaßen gesetzt:

Akkumulator	>	Operand S	Z = 0	CY = 0
Akkumulator	<	Operand S	Z = 0	CY = 1
Akkumulator	=	Operand S	Z = 1	CY = 0

Da der Vergleich einer Subtraktion entspricht, werden ebenfalls das H-Flag, P/V-Flag und N gesetzt.

H: H = 0, bei Übertragung von Bit D4 auf Bit D3, sonst H = 1

S: S = 1, falls das Ergebnis negativ ist, sonst S = Ø

P/V: P/V = 1 bei Überlauf, sonst P/V = 0

N: N = 1 gesetzt.

<u>Beispiele:</u>

Testen Sie, ob auf Speicherplatz FOOOH Bit 4 und Bit 6 gesetzt sind. Ist diese Bedingung erfüllt, soll zusätzlich noch Bit 1 gesetzt werden. (Bit 1 ist vorher Ø oder 1)

```
          LD      HL, ØFOOOH
          LD      A, ßFDH
          AND     M
          LD      B, A
          AND     5OH
          XOR     5OH
          LD      A, B
          JRNZ    MARK1-#
          OR      2
MARK 1:   LD      M, A
END:      HALT
          JR      END - # oder
```

oder	M:	1101 0000	1100 0010
	A:	1111 1101	1111 1101
AND	M	1101 0000	1100 0000
	50 H:	0101 0000	0101 0000
AND	50 H	0101 0000	0100 0000
	50 H:	0101 0000	0101 0000
XOR	50 H	0000 0000	0001 0000

A:	1101 0000
2:	0000 0010
OR2:	1101 0010

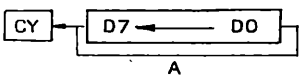
Übung: a) Überprüfen Sie das Beispiel mit Inhalt (HL)= F1H (1111 0001B) und (HL) = 82 H (1000 0010B).

b) Löschen Sie den Inhalt des Akkumulators auf 0 mit Hilfe der Logik-Befehle!

c) Entwickeln Sie ein Programm zur Ermittlung der größten Zahl aus drei positiven Zahlen (1 Byte) und speichern Sie diese auf Speicherplatz F200H. Überprüfen Sie das Ergebnis mit den Zahlen 3, 9, 17.
Die Quelladresse der drei zu testenden Zahlen ist beliebig wählbar.

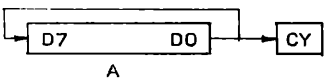
4.4.8. Rotations- und Schiebebefehle

RLCA



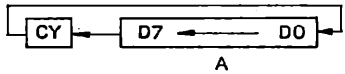
Linkerotation des Akkumulatorinhaltes.
Der Inhalt des Akkumulators wird um eine Bitposition nach links verschoben. Bit D7 wird zum Inhalt von Bit D0 und des Carry-Flags.

RRCA



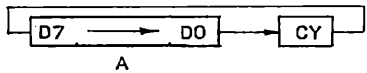
Rechtsrotation des Akkumulatorinhaltes.
Der Inhalt des Akkumulators wird um eine Bitposition nach rechts verschoben. Bit D0 wird zum Inhalt von Bit D7 und des Carry-Flags.

RLA



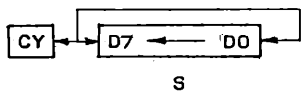
Linksrotation des Akkumulatorinhaltes über das Carry-Flag.
Der Inhalt des Akkumulators wird um eine Bitposition nach links verschoben. Bit D7 wird ins Carry-Flag geschoben und das Carry-Flag ersetzt Bit D0.

RRA



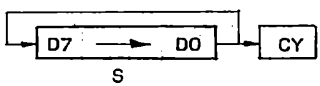
Rechtsrotation des Akkumulatorinhaltes über das Carry-Flag.
Der Inhalt des Akkumulators wird um eine Bitposition nach rechts verschoben. Bit D0 wird ins Carry-Flag geschoben und das Carry-Flag ersetzt Bit D7.

RLC S



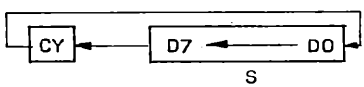
Linksrotation von S analog RLCA.
S: A, B, C, D, E, H, L, (HL), (IX+d),(IY+d)

RRC S



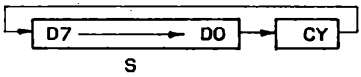
Rechtsrotation von S analog RRCA
S: A, B, C, D, E, H, L, (HL), (IX+d),(IY+d)

RL S



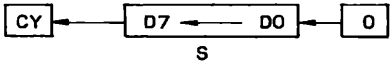
Linksrotation von S analog RLA
S: A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

RR S



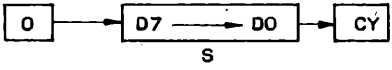
Rechtsrotation von S analog RRA
S: A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

SLA S



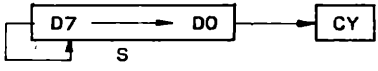
Linksverschiebung von S um 1 Bit. Bit D0 wird 0 und D7 wird zum Inhalt des Carry-Flags
S: A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

SRL



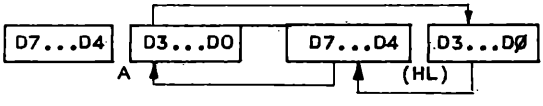
Rechtsverschiebung von S um 1 Bit. Bit D7 wird 0 und Bit D0 wird zum Inhalt der Carry-Flags.
S: A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

SRA



Rechtsverschiebung von S um 1 Bit. Bit D7 bleibt erhalten und Bit D0 wird zum Inhalt des Carry-Flags
S: A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

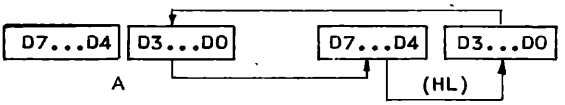
RLD



Verschiebung nach links zwischen dem Akkumulator und dem Inhalt des durch Registerpaar HL adressierten Speicherplatzinhaltes.
Die unteren 4 Bits von (HL) werden in die oberen 4 Bits von (HL) und diese wiederum in die unteren 4 Bits des Akkumula-

tors übertragen. Gleichzeitig erfolgt eine Übertragung der unteren 4 Bits des Akkumulators in der unteren 4 Bit von (HL).

RRD



Verschiebung nach rechts zwischen dem Akkumulator und dem Inhalt des durch Registerpaar HL adressierten Speicherplatzinhaltes.
Die unteren 4 Bits von (HL) werden in die unteren 4 Bit des Akkumulators und diese wiederum in die oberen 4 Bit von (HL) übertragen. Gleichzeitig erfolgt eine Übertragung der oberen 4 Bits von (HL) in die unteren 4 Bit von (HL).

Beispiel:

Umwandlung einer BCD-Zahl in eine DUAL-Zahl
(BCD-Zahl -4Byte- DUAL-Zahl -2Byte)

Lösungsvariante

1. BCD Zahl auftrennen

```
Bsp.:   I 42 I 73 I
        I  4 I  2 I  7 I  3 I
```

2. Ergebnis in DUAL-Zahl umrechnen

```
4x10 = 40
       40 + 2 = 42
                42x10 = 420
                        420+7 = 427
                                427x10 = 4270
                                         4270+3 = 4273
```

3. Ergebnis abspeichern

```
Programm        LD      B,2
                LD      HL, BCDZA+2
                LD      DE, ZWSP+4

UBCD1:          DEC     DE              ,BCD-Zahl auftrennen
                DEC     HL
                XOR     A
                RRD
                LD      (DE), A
                LD      A, M
                DEC     DE
                LD      (DE), A
                DJNZ    UBCD1-//
                LD      H, 0
                LD      L, A
                LD      A, 3
UBCD2:          LD      C, L
                LD      B, H
```

```
        RL      L           ,Multiplikation mit 2
        RL      H
        RL      L           ,Multiplikation mal 4
        RL      H
        ADD     HL, BC      ,Multiplikation mal 5
        RL      L           ,Multiplikation mal 10
        RL      H
        INC     DE
        EX      DE, HL
        LD      C, M
        EX      DE, HL
        LD      B, 0
        ADD     HL, BC      ,Addition der nächsten
                             Zahl
        DEC     A
        JRNZ    UBCD2- //
        LD      (BCDZA), HL

        JMP     END
BCDZA:  BER     2
ZWSP:   BER     4
```

Übung:

a) Überprüfen Sie das Beispiel mit der BCD-Zahl 1281!

b) Schreiben Sie ein Programm, welches den Speicherplatzinhalt von Adresse ADR1 auf gesetzte Bits 2, 4, 7 testet. (Verwenden Sie die Schiebebefehle).
Die Anzahl der gesetzten Bits 2, 4, 7 soll auf Adresse ANZAH abgespeichert werden.

4.4.9. Bitmanipulationsbefehle

SET b, s Die durch b gekennzeichnete Bitposition wird in s gesetzt.
b: 7, 6 ... , 0
s: A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

RES b, s Die durch b gekennzeichnete Bitposition wird in s gelöscht.
b: 7, 6, ..., 0
s: A, B, C, D, E, H, L, (HL), (IX+d), (IY+d)

Bit b, s Die durch b gekennzeichnete Bitposition von s wird komplementiert und ins Zero-Flag Z geladen.

$$Z \longleftarrow s_b$$

b: 7, 6, ..., Ø
s: A, B, C, D, E, H, L, (HL, (IX+d), (IY+d)

Übung:

Schreiben Sie ein Programm, welches den Speicherplatzinhalt von Adresse ADRI auf gesetzte Bits 2, 4, 7 testet. Verwenden Sie dazu die Bitmanipulationsbefehle. Ist die genannte Bedingung erfüllt, soll zusätzlich Bit 1 gesetzt werden.

4.4.10. Blocktransfer- und Suchbefehle

LDI Transport eines Datenbytes von der durch Registerpaar HL adressierten Speicherzelle nach der durch Registerpaar DE adressierten Speicherzelle. Anschließend werden die Register HL und DE um 1 erhöht und Register BC um 1 vermindert.

(HL) ⟶ (DE)
DE ⟶ DE+1
HL ⟶ HL+1
BC ⟶ BC-1

LDIR Transport von mehreren Datenbytes ab der durch HL adressierten Speicherzelle nach der durch DE adressierten Speicherzelle. Die Anzahl der Bytes wird durch das Registerpaar BC festgelegt. Anschließend werden die Register HL und DE um 1 erhöht und Register BC um 1 vermindert bis BC den Wert 0 erreicht.

(HL) ⟶ (DE)
DE ⟶ DE+1
HL ⟶ HL+1
BC ⟶ BC-1

LDD Transport eines Datenbytes von der durch Registerpaar HL adressierten Speicherzelle nach der durch Registerpaar DE adressierten Speicherzelle. Anschließend werden die Register HL, DE und BC um 1 vermindert.

(HL) ⟶ (DE)
DE ⟶ DE-1
HL ⟶ HL-1
BC ⟶ BC-1

LDDR Transport von mehreren Datenbytes ab der durch HL adressierten Speicherzelle nach der durch DE adressierten Speicherzelle. Die Anzahl der Bytes wird durch das Registerpaar BC festgelegt. Anschließend werden die Register HL, DE und BC um 1 vermindert, bis BC den Wert 0 erreicht.

(HL) ⟶ (DE)
DE ⟶ DE-1
HL ⟶ HL-1
BC ⟶ BC-1

CPI Vergleich des Akkumulatorinhaltes mit dem Inhalt der durch HL adressierten Speicherzelle. Entsprechend dem Ergebnis werden die Flags gesetzt. Das Registerpaar BC wird um 1 vermindert und das Registerpaar HL um 1 erhöht.

A ⟷ (HL)
HL ⟶ HL+1
BC ⟶ BC-1

CPIR Vergleich des Akkumulatorinhaltes mit dem Inhalt der durch HL adressierten Speicherstelle. Das Registerpaar BC wird um 1 vermindert und das Registerpaar HL um 1 erhöht. Die Anzahl der zu vergleichenden Bytes wird durch das Registerpaar BC festgelegt. Der Vergleich endet, wenn BC gleich 0 ist bzw. der Akkumulator gleich (HL) ist.

Flags: Z = 1 bei Gleichheit
P/V = 1 falls BC-1=0

A ←→ (HL)
HL → HL+1
BC → BC-1

CPC Vergleich des Akkumulatorinhaltes mit dem Inhalt der durch HL adressierten Speicherstelle. Entsprechend dem Ergebnis werden die Flags gesetzt. Die Registerpaare BC und HL werden um 1 vermindert.

A ←→ (HL)
HL → HL-1
BC → BC-1

CPDR Vergleich des Akkumulatorinhaltes mit dem Inhalt der durch HL adressierten Speicherstelle. Die Registerpaare BC und HL werden um 1 vermindert. Die Anzahl der zu vergleichenden Bytes wird durch das Registerpaar BC festgelegt. Der Vergleich endet, wenn BC gleich 0 ist bzw. der Akkumulator gleich (HL) ist.

Flags: Z = 1 bei Gleichheit
P/V 1 falls BC-1 = 0

A (HL)
HL HL-1
BC BC-1

Beispiel: Laden Sie ab Adresse ANFA die folgenden 100 Byte mit 30H.

```
        LD      HL, ANFA
        LD      M, 30H
        LD      B, 99
        LD      DE, ANFA+1
        LDIR
End:    HALT
        JR      END - #
```

Übung:

a) Überprüfen Sie obiges Beispiel mit dem Lernsystem

b) Schreiben Sie ein Programm, welches einen 100 Byte Block auf die erste mit dem Wert 40H beschriebene Speicherzelle durchsucht.
Ist eine solche Speicherzelle vorhanden, soll ihre Position innerhalb des Blockes auf die Adresse POS abgespeichert werden. Andernfalls ist der Speicherplatz Null mit 1 zu laden.
Überprüfen Sie dieses Programm mit einem selbstgewählten Beispiel.

4.4.11. Spezielle Akkumulator- und Flagbefehle

DAA Der DAA-Befehl korrigiert nach der Addition/Subtraktion zweier gepackter BCD-Zahlen den Inhalt des Akkumulators so, daß wieder gepacktes BCD-Format entsteht.

Beispiel:	0001 1001	1	9
Addition:	0100 0100	4	4
	0101 1101	5	D
nach DAA	0110 0011	6	3

CPL Bitweises Negieren des Akkumulatorinhaltes

Beispiel:		0101 1111
	NEG	1010 0000

NEG Subtraktion des Akkumulatorinhaltes von 0 (entspricht Zweierkomplement)

CCF Komplementieren des CY-Flags

SCF Setzen des CY-Flags

Übung: Addieren Sie die BCD-Zahlen 67 und 15. Überprüfen Sie das Ergebnis mit dem Lernsystem.

4.4.12. Indirekte Registeroperationen

PUSH qq Der Inhalt des Registerpaares qq (AF, BC, DE, HL) oder der Register IX bzw. IY wird in den durch den Stackpointer adressierten RAM-Bereich (Stack) geladen. Dabei wird der höherwertige Teil des Registerpaares in den durch SP-1 adressierten Speicherplatz und der niederwertige Teil in den durch SP-2 adressierten Speicherplatz geladen. Nach Befehlsausführung wird SP-2 zur aktuellen Adresse des Stackpointers.

POP qq Der Inhalt des durch den Stackpointer adressierten Speicherplatzes wird in den niederwertigen Teil des Registerpaares qq und der Inhalt des durch SP+1 adressierten Speicherplatzes wird in den höherwertigen Teil des Registerpaares qq geladen. Nach Befehlsausführung wird SP+1 zur aktuellen Adresse des Stackpointers.
qq: AF, BS, DE, HL, IX, IY

Beispiel:

			PUSH DE
			PUSH HL
aktueller SP:	5FEH		
	5FDH	9Ø H	(HL=9000H)
	5FCH	ØØ H	(DE=7000H)
	5FBH	70 H	
	5FAH	ØØ H	

aktueller SP:	5FAH	ØØ H	POP HL
	5FBH	7Ø	POP BC
	5FCH	ØØ H	(HL=9000H)
	5FDH	9Ø H	(BC=7000H)
	5FEH		

4.4.13. Unterprogrammaufruf- und Rücksprungbefehle

Unterprogramme sind Befehle, die ein- oder mehrmals bei der Abarbeitung eines oder mehrerer verschiedener Programme aktiviert werden. Der Einsprung in ein Unterprogramm (UP) erfolgt über den Befehl CALL und das UP endet mit einem RET-Befehl.

CALL nn unbedingter Unterprogrammaufruf
Der aktuelle Programmzähler (PC) wird im Stack gerettet und anschließend mit Adresse nn geladen.

(SP-1)	PC_H
(SP-2)	PC_L
PC	nn

CANZ nn Unterprogrammaufruf, wenn Z-Flag gleich 0 ist

CAZ nn Unterprogrammaufruf, wenn Z-Flag gleich 1 ist

CANC nn Unterprogrammaufruf, wenn CY-Flag gleich 0 ist

CAC Unterprogrammaufruf, wenn CY-Flag gleich 1 ist

CAPO nn Unterprogrammaufruf, wenn P/V-Flag gleich 0 ist

CAPE nn Unterprogrammaufruf, wenn P/V-Flag gleich 1 ist

CAP nn Unterprogrammaufruf, wenn S-Flag gleich 0 ist

CAM nn Unterprogrammaufruf, wenn S-Flag gleich 1 ist

RST p Der Restart-Befehl RST-p ist ein spezieller Unterprogrammaufruf. Dabei erfolgt ein Sprung zu der angegebenen Adresse p.

p kann sein: 00H
Ø8H
10H
18H
20H
28H
30H
38H

Die damit aktivierte Befehlsfolge muß mit einem RET-Befehl enden.

RET unbedingter Rücksprung

Der Inhalt der durch den Stackpointer adressierten Speicherstelle wird zum aktuellen Programmzähler. Anschließend wird der Stackpointer um 2 erhöht.

PC_L	(SP)
PC_H	(SP+1)
SP	SP+2
RNZ	Unterprogrammrücksprung, wenn Z-Flag gleich Ø ist
RZ	Unterprogrammrücksprung, wenn Z-Flag gleich 1 ist
RNC	Unterprogrammrücksprung, wenn C4-Flag gleich Ø ist
RC	Unterprogrammrücksprung, wenn C4-Flag gleich 1 ist
RPO	Unterprogrammrücksprung, wenn P/V-Flag gleich Ø ist
RPE	Unterprogrammrücksprung, wenn P/V-Flag gleich 1 ist
RP	Unterprogrammrücksprung, wenn S-Flag gleich Ø ist
RM	Unterprogrammrücksprung, wenn S-Flag gleich 1 ist

RETN Rücksprung aus einer Interruptroutine, die durch einen nichtmaskierten Interrupt (NMI) ausgelöst wurde.

Übung:

Schreiben Sie ein Programm, welches die ersten 4 LED's der Anzeige im Abstand von 1 SE nacheinander mit 8 lädt. Benutzen Sie als Zeitschleife ein Unterprogramm.

4.4.14. Allgemeine Steuerbefehle

NOP Die CPU führt keine Operation aus.

HALT Die CPU führt solange NOP-Befehle aus, bis ein Interrupt- oder der RESET-Eingang aktuell wird.

Ein-/Ausgabebefehle

Diese Befehlsgruppe wird ausführlich unter Punkt 5 beschrieben.

5. Programmierung der Peripheriebausteine

5.1. Peripheriespezifische Softwaregrundlagen

5.1.1. Ein-/Ausgabebefehle

- IN n

Operation	A ← (n)	
Maschinencode:	1101 1011	DB H
	n	n

Akkuinhalt wird mit Inhalt des durch n adressierten Eingabekanals geladen.

- IN r

Operation	r ← (c)	
Maschinencode:	1110 1101	EDH
	01 000	

Register r wird mit Inhalt des durch C adressierten Eingabekanals geladen.

-INI

Operation	(HL) ← (C);	B ← B-1;	HL ← HL+1
Maschinencode:	1110 1101	ED H	
	1010 0010	A2 H	

Mit HL adressierte Speicherstelle wird mit dem Inhalt des durch C adressierten Eingabekanals geladen. Anschließend wird B um 1 vermindert und HL um 1 erhöht.

- INIR

Operation	(HL) ← (C);	B ← B-1;	HL ← HL+1
Maschinencode:	1110 1101	ED H	
	1011 1010	B2 H	

wie INI; jedoch Wiederholung der Befehlsausführung bis B = Ø wird.

- IND

Operation	(HL) ← (C);	B ← B-1;	HL ← HL-1
Maschinencode:	1110 1101	ED H	

1010 1010 AA H

wie INI; jedoch wird HL um 1 verringert.

- INDR

Operation (HL) ← (C); B ← B-1; HL ← HL-1

Maschinencode: 1110 1101 ED H

1011 1010 BA H

wie INIR; jedoch wird HL um 1 verringert.

- OUT n

Operation (n) ← A

Maschinencode: 1101 0011 D3 H

n n

Akkuinhalt wird an den Ausgabekanal mit der Adresse n gegeben

- OUT r

Operation (C) ← r

Maschinencode: 1110 1101 ED H

01 r 01

Registerinhalt wird an den durch C adressierten Ausgabekanal ausgegeben.

- OUTI

Operation (C) ← (HL); B ← B-1; HL ← HL+1

Maschinencode: 1110 1101 ED H

1010 0011 A3 H

Der Inhalt, der durch HL adressierten Speicherstelle wird an den durch C adressierten Ausgabekanal ausgegeben. Anschließend wird B um 1 vermindert und HL um 1 erhöht.

- OTIR

Operation (C) ← (HL); B ← B-1; HL ← HL+1

Maschinencode: 1110 1101 ED H

1011 0011 B3 H

wie OUTI; jedoch Wiederholung der Befehlsausführung bis B = 0 wird

- OUTD

Operation	(C) ← (HL);	B ← B-1;	HL ← HL-1
Maschinencode:	1110 1101	ED H	
	1010 1011	AB H	

wie OUTI; jedoch wird hier HL dekrementiert

- OTDR

Operation	(C) ← (HL);	B ← B-1;	HL ← HL-1
Maschinencode:	1110 1101	ED H	
	1011 1011	BB H	

Bemerkung:

Bei den Eingabe- und Ausgabebefehlen liegt die Kanaladresse auf dem unteren Adreßbus A0 ... A7. Auf dem oberen Adreßbus A8 bis A15 liegt bei IN n und OUT n der Akkuinhalt A; bei den Befehlen IN r, OUT r der Inhalt von B; bei den Befehlen INI, INIR, IND, INDR der nicht dekrementierte Wert von B und bei den Befehlen OUT, OTIR, OUTD, OTDR der dekrementierte Wert von B.

Übungsbeispiele zu Abschnitt 5.1.1.

Vorbereitung:

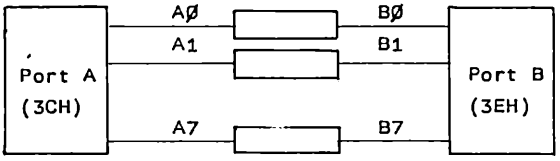
Die Ein-/Ausgabebefehle sollen anhand des PIO-Schaltkreises erkärt werden. Der PIO-Schaltkreis muß dazu entsprechend programmiert werden. Dazu dient das folgende kurze Programm, dessen Bedeutung später erklärt wird.

Geben Sie bitte folgende Bytefolge beginnend bei F 1ØØH ein:

21 ØA F1 Ø1 3D Ø3 ED B3 18 Ø3 CF ØØ Ø7 21 16 F1
Ø1 3F Ø3 ED B3 C9 CF FF Ø7.

Des weiteren stellen Sie bitte in jedem von Ihnen erstellten Programm zu diesem Abschnitt die Bytefolge CD ØØ F1 voran (Aufruf des Initialisierungsprogramm für PIO).

Der PIO-Schaltkreis ist so programmiert, daß der Kanal mit der Adresse 3CH als Ausgabekanal und der Kanal mit der Portadresse 3EH als Eingabekanal arbeitet, wobei beide über Widerstände miteinander verbunden sind.



Der Zustand der Ausgangsleitungen (und damit auch der an Port B anliegende) wird über LED angezeigt. (H = leuchtendes LED).

1. Überprüfen Sie die Wirkung der Befehle OUT n und OUT r. Geben Sie dazu die Bytes 55 H und AAH zur Portadresse 3CH aus. (Startadresse: PC = FØØØH)

1.1. Verwenden Sie dazu den Befehl OUT n.
(STEP-Betrieb)

1.2. Verwenden Sie dazu den Befehl OUT r.
Laden Sie im Programm den Wert 55 H in D und den Wert AAH in E.
(STEP-Betrieb)

2. Überprüfen Sie die Wirkung des Befehls OTIR. Der auszugebende Speicherbereich sei F200 bis F 2ØF.(Füllen Sie diesen Speicherbereich zweimal mit der Bytefolge Ø1, Ø2, Ø4, Ø8, 1Ø, 2Ø, 4Ø, 8Ø) (Startadresse:PC = FØØØH)

2.1. Ausführung mit STEP-Betrieb

2.1. Ausführung mit START-Betrieb

3. Machen Sie sich die Wirkungsweise der Befehle INr klar. Die Register A, D, E sollen in der genannten Reihenfolge mit den Werten Ø1, Ø2, Ø4 geladen werden. Erzeugen Sie diese Werte durch OUTI-Befehle vom RAM F 2ØØH bis F 2Ø2H (s. Aufgabe 2) und lesen Sie diese dann über INr ein!
(Startadresse: PC = F ØØØH)

5.1.2. Interrupts

Die Kommunikation zwischen Mikrorechner und seiner Peripherie beinhaltet zum einen die ausgabeseitige Reaktion des Mikrorechners auf bestimmte eingetretene Bedingungen (Reaktion ist gem. dem rechnerinternen Programm genau festgelegt) und zum anderen die eingabeseitige Erprobung von Kommunikationsanforderungen der Rechnerumgebung. Die Kommunikationsanforderungen (Bedienfunktionen, Dateneingaben, Fertigmeldungen, Alarme) treten abhängig von bestimmten Ereignisabläufen, in der Regel innerhalb bestimmter Zeitgrenzen unabhängig vom Rechnerprogrammlauf auf.
Um diese zu erfassen, muß der Rechner entweder zyklisch alle entsprechenden Signalquellen abfragen (Polling) oder aber die Peripherie meldet selbst durch ein Signal das Auftreten einer Kommunikationsforderung an (Interrupt), unterbricht das laufende Programm und geht in ein Unterprogramm zur Abarbeitung der Forderungen über (Interruptserviceroutine ISR).
Treten Interrupts zeitlich überlappt auf, so wird im U 880 D-System durch die Anordnung der interruptenden Peripheriebausteine in einer Interruptprioritätenkette die Rangfolge der ISR festgelegt.
Das Interruptsystem des U 880 D-System basiert auf zwei getrennten, voneinander unabhängigen Interrupt-Signaleingängen:

- NMI: nicht maskierbarer Interrupt
  INT: maskierbarer Interrupt

Der maskierbare Interrupteingang $\overline{INT}$ kann in 3 verschiedenen Betriebsarten (Interruptmodus) betrieben werden. Diese sind in Abschnitt 2.2.3. näher beschrieben. Im folgenden soll nur auf den Mode 2 eingegangen werden.
Das Einstellen des Interruptmodus 2 erfolgt über den Befehl:

- IM 2

  Maschinencode: 1110 1101 EDH
  0101 1110 5EH

Dieser Befehl legt den Interruptmode 2 fest (nach einem RESET befindet sich die CPU automatisch im IM Ø).
Weitere Interruptsteuerbefehle sind die Befehle EI, DI:

- EI

  Maschinencode: 1111 1011 FBH

Dieser Befehl setzt das Interrupt-Annahme-FF und erlaubt somit die Annahme von Interrupt-Anmeldungen an INT-Eingang.
Die CPU setzt nach jedem RESET und jeder Annahme eines maskierbaren Interrupts automatisch das Interrupt-Annahme-FF zurück (Interrupt gesperrt). Neuerliche Interrupts sind erst nach Freigabe durch EI möglich.

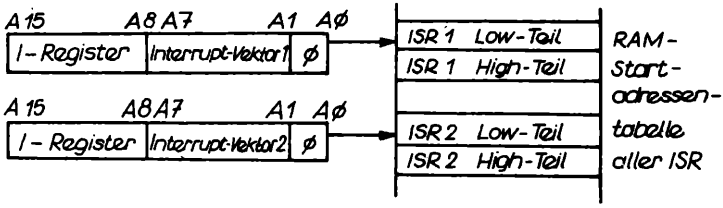
- DI

  Maschinencode: 1111 0011 F3H

Der DI-Befehl sperrt durch Rücksetzen des Interrupt-Annahme-FF den maskierbaren Interrupt.
Programmtechnisch wird dieser Befehl angewandt, um an bestimmten Stellen der Programmabarbeitung die Unterbrechung durch einen Interrupt zu verhindern bzw. eine begonnene ISR nicht durch höherpriorisierte Interrupts zu unterbrechen.

Ein Interrupt-Aufruf im Mode 2 ist vergleichbar mit einem Unterprogrammaufruf CALL. Jedoch wird dieser Aufruf der ISR hardwaremäßig durch Aktivieren des INT-Eingangs der CPU eingeleitet. Da die Peripherie in der Regel zeitlich asynchron

zum CPU-Programmlauf arbeitet, erfolgt diese Unterbrechungsanforderung in den meisten Fällen auch bei einem unbekannten Stand des Programmzählers. Der letzte in Abarbeitung befindliche Befehl des zu unterbrechenden Programms wird abgearbeitet, der interruptende Peripherieschaltkreis sendet seinen Interruptvektor aus, anschließend wird der letzte Programmzählerstand automatisch im Stack abgelegt. Aus dem Interruptvektor (AØ ... A7) und dem Inhalt des I-Registers (A8 A15) wird ein 16-Bit-Zeiger gebildet, der auf die Adresse der jetzt zu beginnenden ISR zeigt.



Da beim Interruptvektor A = Ø ist, muß der LOW-Teil der Startadresse immer auf einer geradzahligen Speicheradresse stehen.

Vor dem Auftreten eines Interrupts müssen somit die Startadressentabelle und das I-Register vom Programmierer definiert geladen werden. Das I-Register läßt sich durch folgenden Befehl beeinflussen:

- LD I, A	Operation: I	A
Maschinencode:	1110 1101	EDH
	0100 0111	47H

Da das Auftreten eines Interrupts einem indirekten Unterprogrammaufruf entspricht, sind auch die dementsprechenden Aktivitäten einzuleiten.
In der Regel wird es notwendig sein, die in der ISR verwendeten Register entweder durch Arbeit mit dem Alternativregistersatz (EXX, EXAF) oder durch PUSH-Befehle zu retten.

Am Ende der ISR ist der Rücktausch (EXAF, EXX) bzw. die Rückkellerung der Register (POP) notwendig.
Der Rückkehr der ISR zum unterbrochenen Programm wird durch den Befehl RETI veranlaßt:

- RETI

  Operation: $PC_L \leftarrow (SP)$; $PC_H \leftarrow (SP+1)$; $SP \leftarrow (SP+2)$

  Maschinencode: 1110 1101 EDH

  0100 1101 4DH

Dieser Befehl bewirkt einen Rücksprung aus einer Interruptroutine.
Der Programmzähler PC wird wie beim Befehl RET mit dem Inhalt des SP geladen. Der Unterschied zum Befehl RET besteht darin, daß der Maschinencode ED4DH hardwaremäßig von den Peripherieschaltkreisen ausgewertet wird, um die zur Zeit höchstpriorisierte in Abarbeitung befindliche ISR zu ermitteln und damit zu beenden.
In der Regel erfolgt der Rücksprung an die Stelle des Programms, wo die Unterbrechung erfolgte. In manchen Fällen ist aber die Rückkehr an eine andere Stelle des Hauptprogramms notwendig. Dazu ist es möglich, eine Stackmanipulation durchzuführen, in dem die automatisch gerettete Rückkehradresse rückgekellert, durch eine neue ersetzt und im Stack abgelegt wird. Eine nachfolgende RETI-Operation bewirkt dann die Rückkehr auf die neue Rückkehradresse.

```
Beispiel:   POP HL          ; Auskellern alte Rückkehr-
                              adresse
            LD HL, RANEU    ; Laden neue Rückkehradresse
            PUSH HL         ; Kellern "    "      "
            RETI            ; Rückkehr zur "
```

Übungsbeispiele zum Abschnitt 5.1.2.

Die programmtechnische Erprobung von Interrupts erfolgt nach Abschnitt 5.2. Folgende Aufgabe dient der theoretischen Festigung der Zusammenhänge:

1. Gegeben sind vier interruptfähige Kanäle A, B, C, D. Die Anfangsadressen der dazugehörigen ISR seien entsprechend FØØ H, F2ØØ H, F3ØØ H, F4ØØ H. Die Interrupttabelle soll beginnend bei F5ØØ H in der Reihenfolge A, B, C, D aufgebaut werden.

1.1. Auf welchen Wert muß das I-Register geladen werden?

1.2. Welche Interruptvektoren müssen die Kanäle A, B, C, D besitzen?

1.3. Wie sieht die Interrupttabelle aus?

Adresse	Wert
F500	?
?	?
...	...

2. In einer Interruptroutine sollen folgende Aktivitäten realisiert werden:
   - Zuweisung von 4 Werten in den 4 folgenden Speicheradressen:

     F3FF : 79 H
     F3FE : 54 H
     F3FD : 5E H
     F3EC : 79 H

   - Ausgabe eines Wertes FE H an die Partadresse 3C H. Schreiben Sie das Programm dazu! (Anfangsadresse PC = FØØØ H)

     Beachten Sie dabei alle notwendigen Aktivitäten (Retten der Register, Interruptfreigabe, Rückkehr vom Interrupt).

5.2. Parallel - Ein-/Ausgabeschaltkreis U 855 D - PIO

5.2.1. Kurzbeschreibung des U 855 D - PIO

Der PIO (Parallel-Input-Output-Controller) dient zur parallelen Ein- und Ausgabe von Daten zwischen CPU und Mikrorechnerperipherie. Beim Datenaustausch zwischen CPU und Peripherie übernimmt der PIO folgende Funktionen:

- Zwischenspeicherung von Ein-/Ausgabe-Daten, sodaß die CPU entlastet und zeitlich von der Peripherie entkoppelt wird
- Realisierung eines Handshake-(Quittungs-)Betriebes bei Datenübergabe und -übernahme
- CPU-unabhängige Überwachung von Peripheriebedienung und Meldung an CPU durch Interrupt
- Interrupterzeugung ebenfalls bei erfolgter Übergabe-/Übernahme-Quittung

Der U 855 D-PIO enthält zwei 8-bit-Ports (Port A und B), die in vier verschiedenen Betriebsarten verwendet werden können:

Mode Ø: Byte-Ausgabe
Mode 1: Byte-Eingabe
Mode 2: Byte-Ein-/Ausgabe (nur Kanal A)
Mode 3: Bit-Ein-/Ausgabe

Die Funktion des PIO wird programmtechnisch, d.h. durch eine Folge von Steuerbytes, festgelegt.

Im Bild 5.1 ist das Blockschaltbild der internen Struktur des U 855 D-PIO dargestellt:

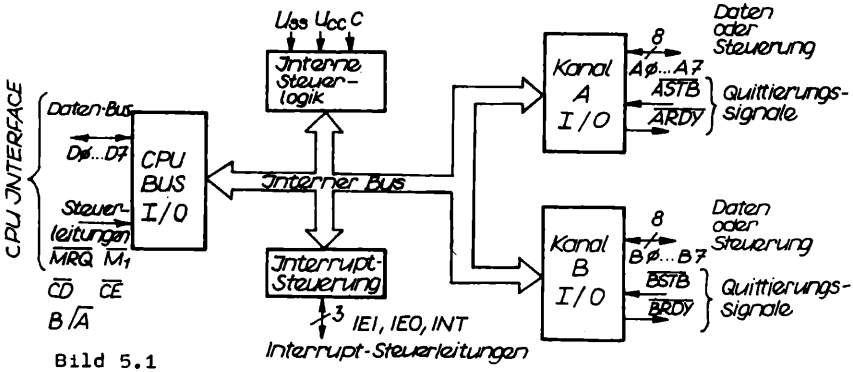


Bild 5.1

Die Daten- und Steuerbus-Anschlüsse ermöglichen eine direkte Zusammenschaltung mit der CPU U 880 D.
Der Datenaustausch (Datenbytes oder Steuerbytes) zwischen PIO und CPU erfolgt mit den unter 5.1. beschriebenen IN/QUT-Befehlen des U 880 D-Befehlssatzes unter Erzeugung der entsprechenden Signalfolgen an den Schaltkreisanschlüssen $\overline{M1}$, $\overline{IORQ}$, $\overline{RD}$, $\overline{CE}$, B/$\overline{A}$, C/$\overline{D}$.
Im U 880 D-System werden die Signale $\overline{CE}$, B/$\overline{A}$, C/$\overline{D}$ in der Regel aus dem Adreßbus der CPU A 7 .. AØ abgeleitet. Dabei aktiviert CE den gesamten PIO-Baustein, BIA wählt Port A oder B aus und C/D kennzeichnet, ob es sich um ein Lesen/Schreiben von Daten oder ein Schreiben von Steuerbytes handelt.

Somit sind für die Adressierung eines PIO-Bausteins vier Portadressen erforderlich:

Port A	Control	(Steuerbyte)	3 DH
Port B	Control	(Steuerbyte)	3 FH
Port A	Daten		3 CH
Port B	Daten		3 EH

Die Interruptsteuerung zwischen PIO und CPU erfolgt über die Interruptsteuerleitung $\overline{INT}$, IEI, IEO. Die $\overline{INT}$-Leitung ist eine Sammelleitung, an welcher alle $\overline{INT}$-Anschlüsse der Peripheriebausteine zusammengefaßt zum $\overline{INT}$-Eingang der CPU geführt werden.

Dabei erfolgt die Anmeldung eines Interrupts über die $\overline{INT}$-Leitung, während die Interruptpriorität bei mehreren Peripherieschaltkreisen durch die Stellung des jeweiligen Schaltkreises in der sogenannten Interruptprioritätenkette bestimmt ist. Die Interruptprioritätenkette wird in der konkreten Mikrorechnerkonfiguration gebildet, indem der Anschluß IEI des nächstniedriger priorisierten Peripherieschaltkreises mit dem Anschluß IEO des nächsthöher priorisierten verbunden wird. Eine laufende, niedriger priorisierte Interruptserviceroutine wird bei Auftreten eines höherpriorisierten Interrupts bis zu dessen vollständiger Abarbeitung unterbrochen. Das folgende Schema soll den Vorgang verdeutlichen:

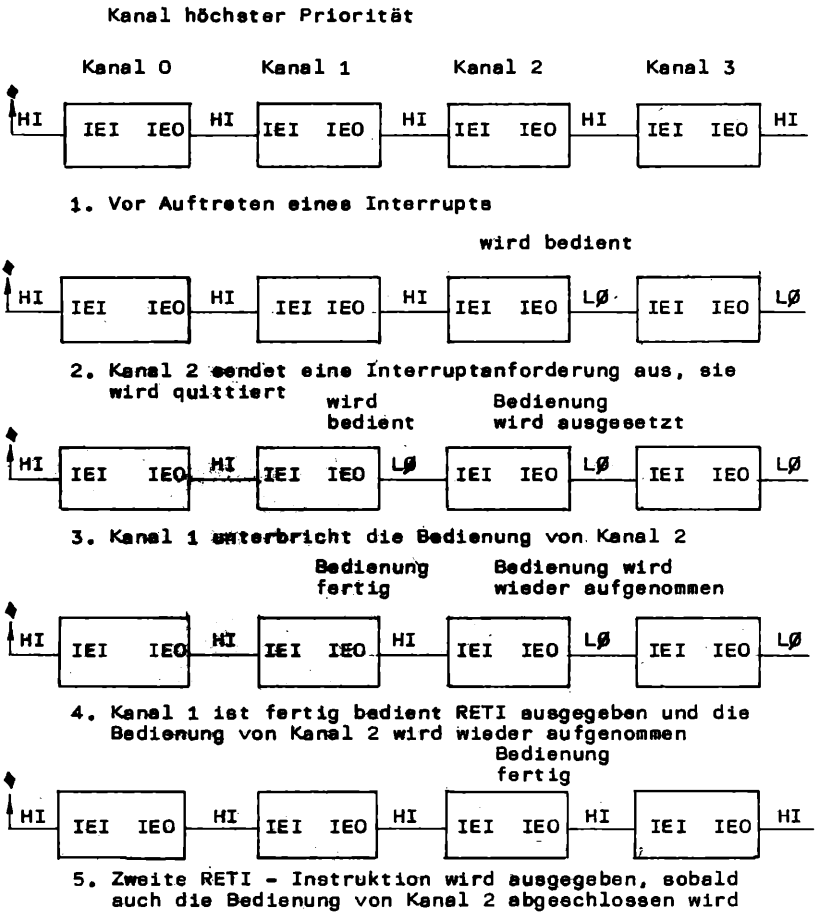


Bild 5.2 Prioritätskaskadierung (Beispiel)

Intern ist im PIO dem Kanal A die höhere Priorität gegenüber Kanal B zugeordnet. Der nähere Ablauf einer Interruptservice-routine wird im Kapitel 6. beschrieben.

5.2.2. Betriebsarten des U 855 D - PIO

- Mode 0 (Byte-Ausgabe Mode)

Der Mode 0 dient der Ausgabe eines 8-bit-Datenwortes von der CPU zur Peripherie. Das Datenwort wird durch einen OUT-Befehl von der CPU zum PIO geschrieben und an den Partleitungen AØ A 7 bzw. BØ ... B 7 bereitgestellt. Der Datenverkehr zwischen PIO und Peripherie erfolgt dann unabhängig von der CPU. Bei Vorhandensein des Datenworts an den PIO-Port-A usgängen erfolgt eine Bereitmeldung am $\overline{ARDY}$ bzw. $\overline{BRDY}$-Ausganges der PIO. Der Abschluß der Datenübertragung (Aktivierung des $\overline{ASTB}$ bzw. $\overline{BSTB}$-Ausganges durch die Peripherie) erzeugt einen Interrupt (genaues Signalspiel siehe Erzeugnisbeschreibung).

- Mode 1 (Byte-Eingabe-Mode)

Der Mode 1 realisiert die Eingabe eines 8-bit-Datenworts von der Peripherie zur CPU. Die Datenübernahme von der Peripherie zum PIO erfolgt unabhängig von der CPU. Durch die Peripherie wird mittels Aktivierung des $\overline{ASTB}$ (BSTB) - Eingangs das Einschreiben des Datenbytes in das PIO-Eingabetor signalisiert. Nach dem Lesen des Bytes durch die CPU (IN-Befehl) zeigt der Ausgang $\overline{ARDY}$ ($\overline{BRDY}$) die erneute Bereitschaft des PIO an, ein Datenbyte zu akzeptieren.
Die Aktivierung von $\overline{ASTB}$ ($\overline{BSTB}$) erzeugt einen Interrupt.
Bei Mode 1 ist zu beachten, daß nach PIO-RESET und nachfolgender Mode 1 - Auswahl von der CPU eine PIO-Leseoperation (Scheineingabe, Datenbyte wird ignoriert) betätigt wird, um das RDY-Signal zu aktivieren.

- Mode 2 (Byte- Ein-/Ausgabemode)

Diese Betriebsart kombiniert die Modes Ø und 1, wobei alle 4 Quittungsleitungen des PIO und die 8 Datenleitungen von Port A verwendet werden.
Die Quittungsleitungen von Port A werden für die Ausgabe-, die Quittungsleitungen von Port B für die Eingabesteuerung

verwendet.
Mode 2 bei Port A erfordert Mode 3 bei Port B.

- Mode 3 (Bit-Ein-/Ausgabe)

Bei Mode 3 werden die Quittungssignale nicht verwendet. Die Ein- und Ausgabe von Daten kann zu jedem beliebigen Zeitpunkt erfolgen. Jedes der 8 Bits des Datenports läßt sich programmtechnisch selektiv als Ein- oder Ausgang definieren. Softwaremäßig definierbare Zustände der einzelnen Bits erzeugen einen Interrupt. Folgende Möglichkeit zur Festlegung einer interrupterzeugenden Bedingung gibt es:

- Bitmaskierung (nur bestimmte Bits werden bewertet)
- Pegelfestlegung (High- oder Low-Pegel der zu bewertenden Bits ist Signifikant)
- Logische Verknüpfung (Signifikante Pegel der maskierten Bits führen AND- oder OR-verknüpft zum Interrupt)

5.2.3. Programmierung des U 855 D-PIO

Die Arbeitsweise des PIO wird durch eine Folge von Steuerworten bestimmt. Die Einstellung der Betriebsart des PIO, die sogenannte Initialisierung, erfolgt nach einem PIO-RESET, kann jedoch auch im Programmlauf beliebig oft geändert werden, wenn der PIO in verschiedenen Betriebsarten benötigt wird. Folgende Aktivitäten beeinflussen die PIO-Funktion:

- RESET (nur hardwaremäßig möglich, Power-On-Reset oder Signalfolge $\overline{M1}$ aktiv bei $\overline{RD}$ und $\overline{IORQ}$ inaktiv)
- Reset der Port-Maskenregister
- Part-Datenleitungen hochohmig, $\overline{RDY}$ inaktiv, automatische Auswahl von Mode 1 (Byte-Eingabe)
- Interruptvektorregister nicht rückgesetzt
- beide Interrupt-Annahme - Flip-Flops- rückgesetzt
- beide Port-Ausgaberegister sind rückgesetzt.

Laden des Interruptvektors

Format:

D7	D6	D5	D4	D3	D2	D1	DØ
V7	V6	V5	V4	V3	V2	V1	Ø

DØ = Ø identifiziert das Steuerwort als Interruptvektor

Laden der Betriebsart

Format:

D7	D6	D5	D4	D3	D2	D1	DØ
M1	MØ	X	X	1	1	1	1

Die Belegung DØ ... D4 = 1 identifiziert das Steuerwort als Mode-Auswahl-Wort

xx - nicht belegt

M1	MØ		
0	0	Mode Ø	(Byte-Ausgabe)
0	1	Mode 1	(Byte-Eingabe)
1	0	Mode 2	(Byte Ein-/Ausgabe)
1	1	Mode 3	(Bit Ein-/Ausgabe)

Ein-/Ausgabe-Selektierung (nur Mode 3)

Nach dem Laden der Betriebsart muß bei Mode 3 das Steuerwort zur Ein-/Ausgabe-Selektierung zum PIO geschrieben werden.

Format:

D7	D6	D5	D4	D3	D2	D1	DØ
I/07	I/06	I/05	I/04	I/03	I/02	I/01	I/0Ø

Laden des Interrupt-Steuerwots

Format:

D7	D6+)	D5+)	D4+)	D3	D2	D1	DØ
Enable interrupt	AND OR	High low	Maske folgt	0	1	1	1

Die mit +) gekennzeichneten Bits (D4 ... D6) sind nur in Mode 3 benutzt.

D7 Ø Interruptflipflop rückgesetzt, Kanal nicht interruptfähig
1 Interruptflipflop gesetzt, Kanal interruptfähig

D6 Ø ODER-Funktion der auf die Erzeugung eines Interrupts einflußnehmenden Kanalleitungen
1 AND-Funktion der auf die Erzeugung eines Interrupts einflußnehmenden Kanalleitungen

D5 Ø Überwachung der auf die Erzeugung eines Interrupts einflußnehmenden Kanalleitungen auf Low-Zustand
1 Überwachung der auf die Erzeugung eines Interrupts einflußnehmenden Kanalleitungen auf High-Zustand

D4 Ø Es folgt keine Maske
1 Maske folgt (s. u.)

Laden der Interrupt-Maske (nur bei Mode 3 nach Interruptsteuerwert mit D4 = 1)

Format:

D7	D6	D5	D4	D3	D2	D1	DØ
MB 7	MB 6	MB 5	MB 4	MB 3	MB 2	MB 1	MB Ø

Dieses Steuerwort dient dazu, diejenigen Kanalleitungen festzulegen, die an der Erzeugung einer Interruptanforderung beteiligt sein können.

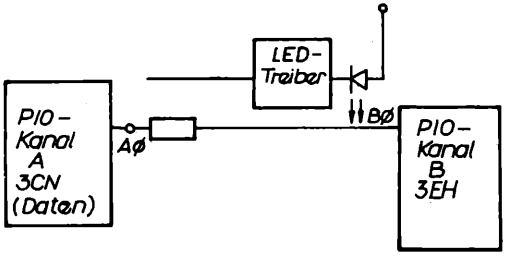
MBn Ø Leitung der Interrupterzeugung beteiligt
1 Leitung von Interrupterzeugung ausgeschlossen.

Übungsbeispiel zum Abschnitt 5.2.

Vorbereitung:

Um den Zustand des vorliegenden PIO-Schaltkreises ein- und ausgangsseitig verfolgen zu können, ist eine Zusatzkarte am Rückseitensteckverbinder vorgesehen.

Sie besitzt folgende Struktur:



Für die Übungen ist vorgesehen, den Kanal A als Ausgabekanal und den Kanal B als Eingabekanal zu betreiben. Der Zustand der Ausgabeleitungen des Kanals A wird über LED angezeigt (H = leuchtendes LED) und entspricht gleichzeitig der am Kanal B anliegenden Eingangsinformation.
Die Adressen des PIO lauten:

3CH	Kanal A Datenwort
3DH	Kanal A Steuerwort
3EH	Kanal B Datenwort
3FH	Kanal B Steuerwort

1.- Schreiben Sie ein Programm zur Initialisierung des PIO-Kanals A mit folgenden Eigenschaften:
Mode 3, alles Ausgänge, kein Interrupt, keine Maske.

- Geben Sie die Information 55H als Datenwort aus.
Startadresse PC = FØØØH

2.- Schreiben Sie ein Programm zur Initialisierung des PIO-Kanals B mit folgenden Eigenschaften:
Mode 3, alles Eingänge, keine Interrupt, keine Maske.

- Lesen Sie die anstehende Eingangsinformation ein und zeigen Sie diese auf der Lernsystemanzeige an.
Zur Anzeige auf dem Lernsystem benutzen Sie bitte die im Betriebssystem des Lernsystems vorhandene Unterroutine DISDAT.
(DISDAT zeigt den Inhalt des Akkus an und wird über CALL DISDAT/Opcode CD DCØ1) aufgerufen).

5.3. Zähler-/Zeitgeberschaltkreis U 857 - CTC

5.3.1. Kurzbeschreibung des U 857 D - CTC

Der U 857 D - CTC ist als Zähler- oder Zeitgeberschaltkreis im Mikrorechnersystem U 880 D vorgesehen. Er besitzt 4 Kanäle, deren Funktion programmtechnisch festgelegt werden kann. Sein Daten- und Steuerbus ist so beschaffen, daß eine direkte Zusammenschaltung mit der U 880 D - CPU möglich ist.

Bild 5.3 zeigt die Struktur des CTC:

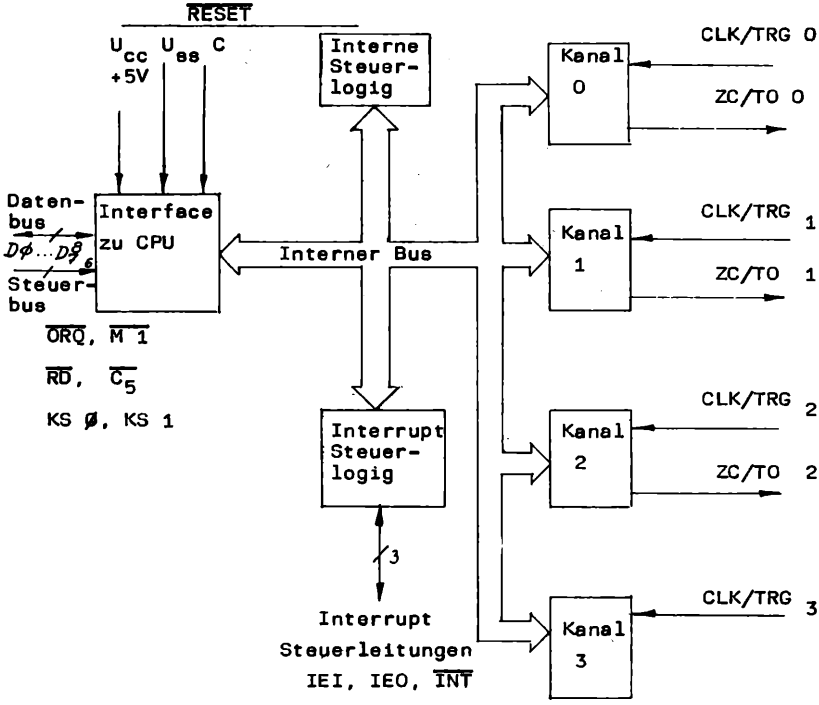


Bild 5.3 Struktur des U 857 D - CTC

Zu den Daten- und Steuerbusanschlüssen sowie den Interrupt-Steuerleitungen des CTC gilt das unter Abschnitt 5.2.1. zum PIO Gesagte.
Abweichend vom PIO besitzt der CTC einen getrennten $\overline{\text{RESET}}$-Eingang.
Nach der Aktivierung des $\overline{\text{RESET}}$ muß der CTC neu programmiert werden.
Die Interruptfähigkeit ist gesperrt. Die Signale KS 1 und KS Ø (meist verbunden mit A1 und AØ der CPU) selektieren im Binärcode einen der 4 Kanäle bei allen Lese-/ oder Schreibzyklen vom / zum CTC. Somit existieren zur vollständigen Adressierung eines CTC-Bausteins vier Portadressen:

CTC-Kanal	Adresse im Lernsystem
CTC Ø	BC H
CTC 1	BD H
CTC 2	BE H
CTC 3	BF H

Im Folgenden ist die Struktur eines CTC - Kanals gezeigt:

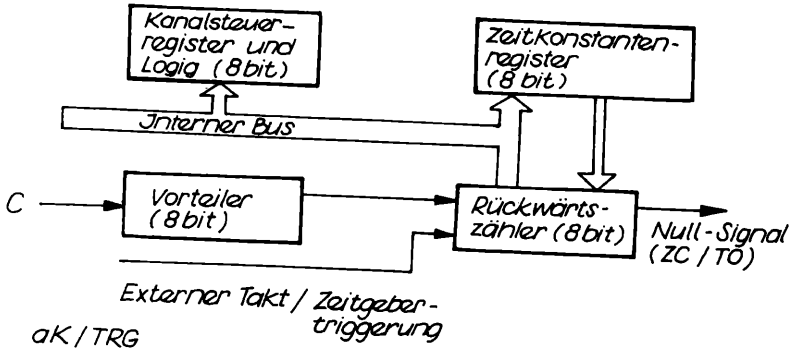


Bild 5.4 Interne Struktur eines CTC - Kanals

5.3.2. Betriebsarten des U 857 D - CTC

Jeder CTC - Kanal kann als Zähler- oder Zeitgeberkanal verwendet werden.
In der Betriebsart Zeitgeber wird der Rückwärtszähler über einen programmierbaren Verteiler (Teilerverhältnis programmierbar auf 16 oder 256) vom Systemtakt dekrementiert.
Der Zeitmeßvorgang kann extern durch einen positiven oder negativen Triggerimpuls als CLK/TRG gestartet werden.
In der Betriebsart Zähler wird der Rückwärtszähler extern durch den CLK/TRG-Eingang dekrementiert. Der momentane Wert des Rückwärtszählers kann zu jedem Zeitpunkt von der CPU gelesen werden.
Jeder Nulldurchgang des Rückwärtszählers kann einen Interrupt erzeugen (wenn freigegeben).
Der 8-Bit-Rückwärtszähler wird mit dem Wert des Zeitkonstantenregister geladen. Das geschieht in beiden Betriebsarten entweder mit dem Laden des Zeitkonstantenregisters durch die CPU oder dann während des Zähler/Zeitgebervorgangs automatisch nach jedem Nulldurchgang des Rückwärtszählers.

5.3.3. Programmierung des U 857 D-CTC

- Laden des Interrupt-Vektors
  Format:

D7	D6	D5	D4	D3	D2	D1	DØ
V7	V6	V5	V4	V3	X	X	Ø

DØ = 0 identifiziert dieses Byte als Interrupt-Vektor.
Der Interrupt-Vektor wird immer an Kanal Ø adressiert.
Die mit X bezeichneten Bits sind im Schreibzyklus ohne Bedeutung, im Interruptfalle ersetzt der jeweils interruptende Kanal die Bits im ausgesandten Interruptvektor durch den Binärwert seiner Kanal-Nr.

- Laden des Kanalsteuerworts
  Format:

D7	D6	D5	D4	D3	D2	D1	D0
0 Interrupt gesperrt 1 Interrupt freigegeb.	0 Betr.-Art Zeitgeber 1 Betr.-Art Zähler	0 Syst.takt um Faktor 16 geteilt 1 Syst.takt um Faktor 265 geteilt	0 Zeitmessung neg. Flanke 1 Zeitmessung pos. Flanke	Triggerzeitpunkt	Zeitkonstante laden	Rücksetzen	1

Nur für Betriebsart "Zeitgeber"

D7 0 Interrupt gesperrt

1 Interrupt freigegeben: Int.anforderung erfolgt jedesmal, wenn der Rückwärtszähler den Wert Null erreicht

D6 0 Zeitgeber: Rückwärtszähler wird vom Vorteiler gestartet
Periode des Zählers $c = t_c \cdot p \cdot TC$

System-periode — Vorteiler (16 od. 256) — 8bit Zeitkonstante (max. 256)

D5 s. o.

D4 s. o.

D3 0 Wenn bit2 = 0 beginnt d. Zeitgeber d. Zeitmessung am Anfang des nächsten Maschinenzyklus
Wenn bit2 = 0 beginnt d. Zeitgeber d. Zeitmessung am Anfang des Maschinenzyklus, der auf das Laden der Zeitkonstante erfolgt.

1 Wenn bit2 = 0 beginnt d. Zeitgeber d. Zeitmessung nach d. lfd. Maschinenzyklus, sobald am Triggereingang die vorgesehene Flanke auftritt.
Wenn bit2 = 1 beginnt d. Zeitgeber d. Zeitmessung nach d. Laden d. Zeitkonstante, sobald am Triggereingang die vorgesehene Flanke auftritt.

D2 0 Auf das Kanalsteuerwort folgt keine Zeitkonstante, d. Zeitkonstante ist zum Anlaufen lassen des Zeitmeßvorganges noch einzugeben.

1 nächstes Kontrollwort für den betreffenden Kanal stellt Zeitkonstante für den Rückwärtszähler dar. Wird während des Zeitmeßvorganges eine neue Zeitkonstante eingegeben, so wird die alte Messung erst zu Ende geführt.

D1 0 Kanal zählt weiter

1 Kanal unterbricht den Zähler bis neue Zeitkonstante eingegeben wird. Ausgang ZC/TO ist inaktiviert und der Kanal kann keinen Interrupt anfordern.

D = 1 indentifiziert das Byte als Kanalsteuerwort

- Laden der Zeitkonstante

  Format:

D 7	D 6	D 5	D 4	D 3	D 2	D 1	D Ø
ZK7	ZK6	ZK5	ZK4	ZK3	ZK2	ZK1	ZKØ

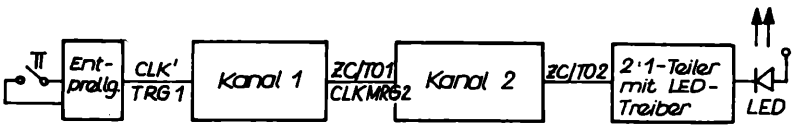
Die Zeitkonstante wird grundsätzlich nach einem Kanalsteuerwort mit D 2 = 1 geladen.
Eine Zeitkonstante ØØH wird als Zeitkonstante 256 interpretiert.

Übungsbeispiele zum Abschnitt 5

(Abschlußübung zur Programmierung von Peripheriebausteinen)

Vorbereitung:

Bei den nachfolgenden Übungen soll mit dem CTC-Kanälen 1 und 2 gearbeitet werden. Die Kanäle sind wie folgt verschaltet:



Die Adressen der Kanäle lauten:

BCH	Kanal 0
BDH	Kanal 1
BEH	Kanal 2
BFH	Kanal 3

1. Erstellen Sie ein Programm mit folgendem Inhalt:
   - Initialisierung des CTC als Zähler, Zeitkonstante OAH, kein Interrupt.
     (Anfangsadresse PC = FØØØH)

- Zyklisches Abfragen des Rückwärtszählerinhalts und Anzeige auf der Lernsystemanzeige
  (Hinweis zur Anzeige: siehe 2. Übungsaufgabe PIO)
- Betätigen Sie den Mikrotaster!

2. Kanal 1 soll als Zeitgeber mit der maximal möglichen Zeit betrieben werden.
   - Ermitteln Sie die maximale Periode eines Kanals als Zeitgeber!
   - Programmieren Sie Kanal 1 als Zeitgeber mit maximaler Periode!
     (Programmbeginn PC = FØ2Ø)

3. Kanal 2 soll als Zähler in Kette mit Kanal 1 (Zeitgeber) betrieben werden. Die resultierende Periode soll 1 s betragen.
   - Ermitteln Sie die dazu notwendige Zeitkonstante!
   - Programmieren Sie Kanal 2 als Zähler mit der ermittelten Konstanten!
     (ohne Interrupt)
   - Schließen Sie dieses Programm nach dem Eingeben von 4 NOP's
     (Lehrbefehl, keine Operation, Opcode ØØH) an das Programm von Aufgabe 2 an!
   - Schließen Sie das Programm durch einen HALT-Befehl ab und starten Sie es!

4. - Ändern Sie die Programmierung von Kanal 2 auf Zählerbetrieb mit Interrupt! Ersetzen Sie den HALT-Befehl durch einen absoluten Sprung auf Adresse FØ4ØH!

     Beachten Sie bitte das Laden des Interruptvektors!
     Fügen Sie das an Stelle der 4 NOP's ein!
     (Die Interrupttabelle soll auf der Speicherstelle FØ5ØH beginnen)
   - Schaffen Sie die programmtechnischen Voraussetzungen für die Arbeit mit einem Interrupt!

1. Schaffung der CPU-Bedingungen (Interruptmode, Interruptfreigabe, I-Register); Programmbeginn PC = FØØØH

2. Belegen der Interrupttabelle mit den Werten.
   (Die ISR soll auf Adresse F 1ØØH beginnen)

3. In der Interruptroutine soll mit dem PIO-Kanal A als Ausgabekanal gearbeitet werden. Dieser muß daher vor dem Auftreten eines Interrupts programmiert werden. Schreiben Sie dazu ein Programm als Speicherstelle FØØ7 (Anschluß an Programmteil unter Punkt 1). Hinweis dazu:
   Übernehmen der Initialisierung aus PIO-Übung Aufgabe 1.
   Beenden Sie diesen Programmteil durch einen absoluten Sprung auf Adresse FØ2ØH.

5. Das zu unterbrechende Hauptprogramm sei eine endlose HALT-Schleife:

```
M:   HALT
     JRM
```

Der Aufenthalt in der HALT-Schleife läßt sich durch das Aufleuchten des HALT-LED's am Lernsystem verfolgen.
Vor der HALT-Schleife soll noch der Akten definiert auf Ø 1 gesetzt werden. Programmbeginn PC = FØ4ØH

6. In dieser Aufgabe soll die Interruptserviceroutine realisiert werden. In der ISR sind folgende Aktivitäten abzuarbeiten:
   Der im Aufgabenteil 5 geladene Akku soll zum PIO-Kanal A (Portadresse 3CH) ausgegeben werden und das gesetzte Bit um eine Stelle weitergerückt werden. Bei ständigen Aufruf der ISR wird dann das leuchtende LED jeweils eine Stelle weiterwandern.
   Beachten Sie bitte alle notwendigen Aktivitäten in der Interruptroutine (EI, RETI)!
   Das Interruptprogramm soll auf der Adresse F1ØØH beginnen.

7. (Zusatzaufgabe)

In dieser Aufgabe soll das Auftreten zweier Interrupts demonstriert werden.
Dieser zweite Interrupt soll durch den PIO-Kanal B ausgelöst werden. Dazu soll dieser auf Bit-Eingabe initialisiert werden und bei Auftreten von High-Pegel an Bit D7 interrupten. Der PIO-Kanal B ist niedriger priorisiert als der CTC (bedingt durch Anordnung in der Prioritätenkette).

- Schließen Sie das PIO-Initialisierungsprogramm an das Programm von Aufgabe 4.3. an! Beachten Sie dabei bitte das Laden des Interruptvektors!

- Erweitern Sie die Interrupttabelle als Adresse FØ5Ø! (Die PIO-ISR soll als Adresse F2ØØH beginnen)

- Schreiben Sie die ISR mit folgendem Inhalt:

  Zuweisen von 4 Werten zu den Speicherplätzen:

F3FF:	79 H
F3FE:	54 H
F3FD:	5E H
F3FC:	79 H

  Uminitialisierung des PIO-Kanals B (Interruptsperre)

6. PROGRAMMBEISPIELE

Die folgenden Programmbeispiele sollen zum Verständnis der Wirkungsweise verschiedener U 880-Befehle dienen. Universelle Programme sind als Unterprogramme ausgeführt. Die dargestellten Beispiele sollen typische Verfahrensweisen z. B. der Arithmetik deutlich machen.
Bei allen Programmbeispielen ist vor dem Start bzw. schrittweisem Test der Programme der Stackpointer zu setzen. Es ist empfehlenswert, den SP jeweils auf F3ØØH zu setzen.

1. Beispiel Zeitverzögerung

- Varianten: Zeitschleife als Zählschleife
  Zeitverzögerung mittels repetierender Befehle

```
OP-Code          Mnemonik                  Kommentar
Ø6 FF            LD B,  LANG1              ;Länge äußere Schleife
ØE ØØ    M1:     LD C,  LANG2              ;Länge innere Schleife
ØD       M2:     DEC C
2Ø FD            JR NZ, M2-#               ;innere Schleife,
                                            6,4 µs je Durchlauf
1Ø F9            DJNZ   M1-#
C9               RET
```

repetierender Befehl:

```
OP-Code          Mnemonik                  Kommentar
Ø1 ØØ 1Ø         LD BC,  LAENG             ;Länge der Verzögerung
21 ØØ FØ         LD HL,  RAMX              ;beliebiger RAM-Speicherplatz
E5       MSEK:   PUSH HL
D1               POP DE
ED BØ            LDIR                      ;8,4 µs je Zyklus
C9               RET
```

Der Programmteil MSEK ist ein Unterprogramm des Lernsystem-Monitorprogramm. Er ist vom Anwender nutzbar. Zu beachten ist, daß das Registerpaar HL keinen ROM-Speicherplatz adressiert, um ein Beschreiben von ROM-Schaltkreisen zu vermeiden.

Das Unterprogramm MSEK realisiert eine Zeitverzögerung von 8,4 us je LDI-Befehlszyklus.

2. Beispiel BCD-7-Segment-Wandlung

Um eine BCD-Ziffer bzw. eine Hexadezimalzahl mittels einer 7-Segment-Anzeige (z. B. VQB71, VQE24 usw.) auszugeben, gibt es zwei Möglichkeiten:

1. BCD-7-Segment-Wandlung mittels Dekoder-Schaltkreis (D146, D147)

2. softwaremäßige BCD-7-Segmentwandlung

Die 2. Variante bietet den Vorteil, andere Kodierungen als die Ziffernbelegung auszugeben. Eine übliche Methode ist die Umwandlung mittels einer Segmenttabelle.

Eintrittsbedingung: BCD-Einzelziffer in Register L
(Bit Ø ... Bit 3, Bit 4 ... Bit 7 = Ø)

Op-Code	Mnemonik	;Kommentar
26 ØØ	LD H, Ø:	;Register H löschen
11 B9 Ø7	LD DE, SEGTA	Anfangsadresse Segmenttabelle
19	ADD HL, DE	;Speicherplatz des 7-Segment-Kodes
7E	LD A, M	
21 FB F3	LD HL, ANZSP	;Anzeigespeicher
77	LD M, A	;Anzeige laden
C9	RET	

Dieses Programm wandelt eine BCD-Einzelziffer in den 7-Segment-Kode und bringt diesen auf der niederwertigen Datenstelle zur Anzeige. Verwendet wird dabei die Segmenttabelle des Lernsystems, womit alle Ziffern (0 ... 9, A ... F) dargestellt werden können.

3. Beispiel: Zählschleife auf Lernsystem-Anzeige

Sehr oft steht die Aufgabe, zyklische Vorgänge zu bearbeiten und den jeweiligen Stand optisch zu signalisieren. Im folgenden Beispiel soll eine Zählschleife realisiert werden und der Stand in der Anzeige erscheinen. Zur Zeitverzögerung wird das im Lernsystem benutzte Unterprogramm MSEK verwendet.

Op-Code		Mnemonik	Kommentar
		ORG ØFØØØH	
3E ØØ		LD A, ØØ	;Anfangswert
11 B9 Ø7	M1:	LD DE, SEGTA	;Anfang Segmenttabelle
26 ØØ		LD H, ØØ	;Register H löschen
6F		LD L, A	;Wert in Register HL
19		ADD HL, DE	
4E		LD C, M	;7-Segment-Kode in Register C
21 FA F3		LD HL, ANZSP	;Anzeigespeicher
71		LD M, C	
Ø1 ØØ FF		LD BC, LAENG	;Geschwindigkeit
21 ØØ F1		LD HL, RAMX	;beliebige Zeile
CD D7 Ø1		CALL MSEK	;Zeitverzögerung
C6 Ø1		ADD1	;Zähler erhöhen
27		DAA	;Dezimalkorrektur
E6 ØF		AND ØFH	;obere 4 Bit löschen
C3 Ø2 FØ		JMP M1	

4. Beispiel: Laufschrift

Die 7-Segment-Anzeige ist mit Einschränkung außer zur Zahlendarstellung auch zur Ausgabe von alphanumerischen Zeichen geeignet.
Hierbei muß eine möglichst gut erkennbare Segmentbelegung für die jeweiligen Zeichen gewählt werden. Dieses Programmbeispiel soll eine Zeichenkette über die Lernsystemanzeige als Endlosband ausgeben.

```
Op-Code                Mnemonik              Kommentar
                       ORG ØFØØØH
21 Ø5 F1      M1:      LD HL, TEXT+5         ;Textzeiger auf Text-
                                               anfang
E5            M2:      PUSH, HL
11 FA F3               LD DE, ØF3FAH         ;rechte Anzeige
Ø6 Ø6                  LD B, Ø6
7E            M3:      LD A, M
12                     LD (DE), A
2B                     DEC HL                ;eine Stelle nach links
13                     INC DE
1Ø FA                  DJNZ M3 -#            ;6 Stellen
E1                     POP HL
23                     INC HL                ;Textzeiger
54                     LD D, H
5D                     LD E, L               ;DE = HL
Ø1 ØØ ØØ               LD BC, Ø
ED BØ                  LDIR                  ;Zeitverzögerung
3E FF                  LD A, ØFFH            ;Endmarke
BE                     CP M                  ;Ende erreicht?
CA ØØ FØ               JPZ M1-#              ;wenn Textende
C3 Ø3 FØ               JMP M2
```

Der Text befindet sich für dieses Beispiel im RAM-Bereich ab der Adresse ØF1ØØH. Die ersten 6 Zeichen und die letzten 6 Zeichen müssen Leerzeichen sein (ØØH). Als Endmarke wird die Zahl ØFFH eingetragen.

5. Beispiel: 4-Byte-Addition im Speicher

Die arithmetischen Operationen werden im Binärsystem analog wie im Dezimalsystem ausgeführt, Bei übertragsbehafteten Operationen muß der entsprechende Übertrag bei Bedarf entweder gerettet bzw. darf nicht durch andere Befehle beeinflußt werden.
Das folgende Beispiel addiert zu einer 4-Byte-Zahl im RAM-Speicher eine weitere 4-Byte-Zahl.

Operation:

(HL+3)(HL+2)(HL+1)(HL)+(DE+3)(DE+2)(DE+1)(DE)=(HL+3)(HL+2)(HL+1)(HL)

Op-Code		Mnemonik	Kommentar
21 ØØ F1		LD HL, ZAHL1	;1.Summand und Ergebnis ;ab F1ØØH
11 04 F1		LD DE, ZAHL2	;2. Summand ab F1Ø4
Ø6 Ø4		LD B, 4	
97		SUB A	;CY löschen
1A	M1:	LD A, (DE)	;2. Summand
8E		ADC M	;Addition
77		LD M, A	
23		INC HL	;nächstes Byte
13		INC DE	
1Ø F9		DJNZ M1-#	
C9		RET	

6. Beispiel: 8-Bit-Multiplikation

Die CPU U 880 führt keine Multiplikation direkt aus. Deshalb muß die Multiplikation auf Additions- und Verschiebebefehle zurückgeführt werden. Prinzipiell bestehen zwei Möglichkeiten:

- n-mal Addition
- stellenweise Addition und Verschiebung

Die erste Variante ist einfach zu programmieren, aber umständlich in der Abarbeitung. Ein weiterer Nachteil besteht darin, daß die Abarbeitungszeit sehr unterschiedlich ist. Die zweite Variante arbeitet analog der dezimalen Multiplikation. Es erfolgt eine stellenweise Addition und entsprechende Verschiebung.

Prinzip:

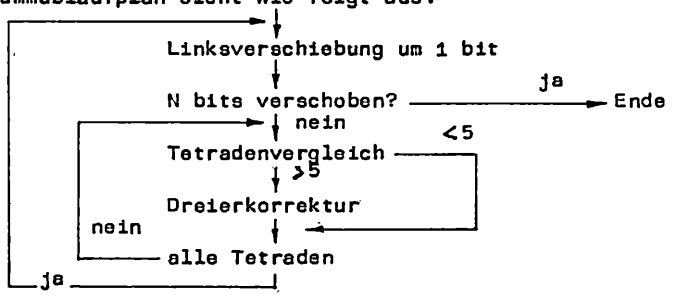
```
  10110 x 101      (22x5)
      10110
+    00000
+     10110
=   1101110      (=110)
```

Programm: C x D = BC

Op-Code	Mnemonik	Kommentar
	MULT:	
97	SUB A	;Register A löschen
Ø6 Ø8	LD B, 8	
	MULT1:	;Anzahl der Bit je ;Multiplikend
CB 19	RR C	
3Ø 01	IRNC MULT2- #	
82	ADD D	
	MULT2:	
CB 1F	RRA	
1Ø F7	DJNZ MULT1- #	
CB 19	RR C	
47	LD B, A	
C9	RET	

7. Beispiel: Binär-BCD-Wandlung

Die Verarbeitung der Daten erfolgt im Mikroprozessor hauptsächlich binär. Für Datenausgaben über Anzeigen, Drucker usw. wird meist die BCD-Darstellung benötigt. Zur Umwandlung von Binärzahlen in BCD-Zahlen gibt es verschiedene Verfahren. Die hier vorgestellte Variante arbeitet nach dem Prinzip der bedingten Dreierkorrektur. Die umzuwandelnde Zahl wird bitweise nach links in einen vorher gelöschten Speicher verschoben. Nach jeder Verschiebung erfolgt in den Tetraden, welche größer gleich 5 sind, eine Addition von 3. Der Programmablaufplan sieht wie folgt aus:



Die binäre Abarbeitung soll das folgende Beispiel verdeutlichen:

Binärzahl: Register D = 63 H = 99 D

BCD-ZIELSPEICHER: Register = = Ø

Aktivität	E	D
	00000000	01100011
1.Verschiebung	00000000	1100011
2.Verschiebung	00000001	100011
3.Verschiebung	00000011	00011
4.Verschiebung	00000110	0011
Korrektur(+Ø3H)	00001001	0011
5.Verschiebung	00010010	011
6.Verschiebung	00100100	11
7.Verschiebung	01001001	1
Korrektur(+Ø3H)	01001100	1
8.Verschiebung	10011001	
	9 9	

Als BCD-Ergebnis erhält man in Register E die Dezimalzahl 99.

Das entsprechende Programm:

Eintritt: Binärzahl in Register D
Zielspeicher Register E

```
Op-Code          Mnemonik      Kommentar

97               SUB A         ; Register A löschen
5F               LD E,A        ; Register E löschen
Ø6 Ø8            LD B,8        ; Datenlänge 8 bit
        KONV1:
CB 12            RL D          ; Verschieben
CB 13            RL E
Ø5               DEC B         ; letzte Verschiebung ?
C8               RET Z         ; Ende, wenn letzte Verschie-
                               ; bung
7B               LD, A,E       ; Zielspeicher
E6 ØF            AND ØFH       ; obere Tetrade abtrennen
```

```
FE Ø5          CP Ø5          Test auf kleiner 5
7B             LD, A,E
38 Ø2          JR, C, KONV2-#  ; Sprung, wenn Tetrade
                               ; kleiner 5
C6 03          ADD Ø3         ; Korrektur untere Tetrade
      KONV2:
FE 5Ø          CP 5Ø H        ; Test obere Tetrade
38 Ø2          JR C, KONV3- #  ; Sprung, wenn kleiner 5Ø
C6 3Ø          ADD 3Ø H       ; Korrektur obere Tetrade
      KONV3:
5F             LD E,A         ; korrigierter Wert in
                               ; Zielspeicher
18 E7          JR KONV1-#
```

8. Beispiel:

Sehr kurz lassen sich Programme erstellen, wenn man auf entsprechend geeignete Unterprogramme zurückgreifen kann. Der Anwender des Lernsystems hat die Möglichkeit, die vorhandene Tastatur in seine Programme einzubeziehen. Der Programmteil TABFR realisiert die Ansteuerung, Abfrage usw. der Tastatur. Bei Austritt aus diesem Unterprogramm liegt in Register A die Tastenwertigkeit der gedrückten Taste. Das folgende Beispiel soll zeigen, daß nur wenige Befehle notwendig sind, um die Wertigkeit jeder beliebigen Taste (außer Reset) zu ermitteln und auf der vorhandenen Anzeige auszugeben.

```
Op-Code        Mnemonik       Kommentar

CD 55 Ø1       CALL TABFR     ; Tastaturabfrage
CD DC Ø1       CALL DISDAT    ; Datenanzeige laden
76             HALT
```

Um die Reaktion des Programmes optisch wirksamer werden zu lassen, soll zusätzlich vor der Tastatur die Anzeige abgeschaltet und nach dem Laden der Anzeige wieder eingeschaltet werden.

Op-Code	Mnemonik	Kommentar
CD 4Ø Ø1	CALL DISOFF	;Anzeige abschlaten
CD 55 Ø1	CALL TABFR	;Tastaturabfrage
CD DC Ø	CALL DISDAT	;Datenanzeige laden
CD 38 Ø1	CALL DISON	;Anzeige einschalten
76	HALT	

9. Beispiel:

Um anfängliche Schwierigkeiten und Fehler bei der Programmierung eines interruptfähigen Systems zu verringern, wird nochmals kurz eine mögliche Vorgehensweise dargestellt:

- Wahl des Interrupt-Mode (IM2)
  Laden des Interrupt-Vektor-Registers der CPU (I-Register)
- Laden der Interrupt-Vektoren der interruptfähigen Peripheriekanäle
- Initialisierung der Peripheriekanäle
- Interruptfreigabe (EI)

Innerhalb einer Interrupt-Routine muß erneut eine Interrupt-Freigabe (EI) erfolgen, da der Interrupt-Annahme-Zyklus automatisch weitere Interrupts (auch höher priorisierter Kanäle) sperrt (entsprechend DI).
Erfolgt der Eintritt in eine Interrupt-Routine innerhalb eines anderen Programmes, müssen die Inhalte der Register gerettet werden, welche innerhalb der Interrupt-Routine beeinflußt werden. Das Retten der Register kann mittels der EXX- bzw. EX AF, AF'-Befehle oder der PUSH und POP-Befehle erfolgen. Üblich ist die Verwendung der Stackoperationen PUSH und POP.

Eine typische Routine, in welcher die Register A, F, B, C, D, E, H und L verwendet werden, sieht wie folgt aus:

```
INTUP:          Interrupt-Unterprogramm
PUSH AF
PUSH BC
PUSH DE
PUSH HL


 .
POP HL
POP DE
POP BC
POP AF
EI
RETI
```

Auf der Basis eines CTC-Interrupts soll eine Zählschleife aufgebaut werden. Der Inhalt des Registers A wird je Sekunde um eins erhöht.
Der Inhalt des Registers A soll über das PIO-Port A ausgegeben und im 7-Segment-Code auf der Datenanzeige sichtbar werden.
Im folgenden Programmbeispiel wird der CTC-Kanal 0 auf interruptfähigen Zeitgeber programmiert. Das PIO-Port A wird auf Mode 3 initialisiert und alle Bits auf Ausgabe geschaltet. In der Interrupt-Routine (INTPR) erfolgt das Decrementieren des BC-Registerpaares. Bei Erreichen des Standes BC = 0 wird dieses Doppelregister mit dem Wert 1000 geladen, das Register A incrementiert und über das PIO-Port A ausgegeben und die Datenanzeige entsprechend geladen. Da der CTC-Kanal 0 im Zeitabstand con ca. 1ms interruptet arbeitet die Zählschleife im Sekundentakt.

Das entsprechende Programm:

Loc.	Op-Code	Mnemonik	Kommentar
		ORG ØFØØØH	
FØØØ	ED 5E	IM2	;Vektorinterrupt
FØØ2	3E F1	LD A, ØF1H	;Interruptvektor High- ;Byte
FØØ4	ED 47	LD I,A	;Interruptvektorregi- ;ster laden
FØØ6	3E ØØ	LD A,Ø	;Interruptvektor für ;CTC.
FØØ8	D3 BC	OUT CTCØ	
FØØA	3E 85	LD A, 85H	;Zeitgeber mit INT, ;Teiler 16
FØØC	D3 BC	OUT CTCØ	
FØØE	3E 9C	LD A, 9CH	;Zeitkonstante
	D3 BC	OUT CTCØ	
FØ12	3E CF	LD A, ØCFH	;PIO-Mode 3
FØ14	D3 3D	OUT OIPAC	
FØ16	3E ØØ	LD A, Ø	;E/A-Festlegung alle ;Bits Ausgabe
FØ18	D3 3D	OUT OIPAC	
FØ1A	3E Ø7	LD A,7	;kein INT
FØ1C	D3 3D	OUT OIPAC	
FØ1E	AF	XOR A	;Register A löschen
FØ1F	Ø1 E8 Ø4	LD BC, 4 E8H	;Zähler 1000 msec.
FØ22	FB	EI	;INT-Freigabe
FØ23	76	M1O:HALT	
FØ24	18 FD	R M1Ø-#	
		INTPR:	;Interrupt-Programm
FØ26	ØD	DEC C	
FØ27	2Ø Ø6	JR NZ, M2Ø-#	
FØ29	1Ø Ø4	DJNZ M2Ø-#	
FØ2B	Ø1 E8 Ø3	LD BC, 3E8H	
FØ2E	3C	INC A	
FØ2F	D3 3C	M2O: OUT OIPAD	
FØ31	CD DC Ø1	CALL DISDAT	

Loc.	Op.-code	Mnemonik	Kommentar
FØ34	FB	EI	
FØ35	ED 4D	RETI	
		ORG ØF1ØØH	
		INTTAB:	;Interrupt-Tabelle
F1ØØ	26 FØ	DEFW INTPR	;Startadresse des ;Interrupt-Programms

10. Beispiel: U h r

Ein bei der Nutzung von Rechnern oft auftretendes Problem ist die Erstellung von Uhrenfunktionen. Dieses trifft zum Beispiel für zeitabhängige Steuerungen, Zeitmessungen oder Protokollausdrucke mit Angabe der Uhrzeit zu.

Die Uhrenfunktion kann dabei in verschiedene Teilprobleme aufgegliedert werden.

a) Zeitbasis

Für die Funktion der Uhr ist die Bereitstellung einer entspr. genauen Zeitbasis notwendig. Prinzipiell bietet sich dafür der Systemtakt des Mikrorechners an, da er in der Regel von einem Schwingquarz abgeleitet wird. Im einfachsten Fall, d. h. mit minimalem Hardware-Aufwand, kann daraus durch eine programmierte Zeitschleife ein Uhrengrundtakt erzeugt werden. Dabei sind alle während eines Uhrenzyklus auftretenden Takte mit der systemspezifischen Taktzeit (im Fall des Lernsystems 0,4 µs) zu multiplizieren.

Beispiel:

```
                LD B, Ø          7 Takte
MARKE:          DJNZ MARKE-#     13 Takte, wenn B ≠ Ø,sonst
                                 8 Takte
```

In diesem Beispiel wird das B-Register mit einer Zahl, die die Anzahl der Schleifenläufe angibt, geladen. An-

schließend wird das B-Register durch den Befehl DJNZ so lange dekrementiert, bis es den Wert Ø erreicht hat. Die Abarbeitung benötigt

(7 + 255 13 + 8) • 0,4 µs = 1,332 ms.

Dieses ist die maximale Verzögerungszeit, welche mit dieser Verzögerungsschleife erzielt werden kann. Durch Einfügung von Leerbefehlen kann die Zeit erhöht werden,

```
        LD B, Ø          7 T
MARKE:  NOP              4 T
        DJNZ MARKE-#     13/8 T
```

ebenso durch eine Verschachtelung mehrerer Schleifen.

```
        LD C, Ø          7 T
MARK1:  LD B, Ø          7 T
MARK2:  DJNZ MARK2-#     13/8 T
        DEC C            4 T
        JRNZ MARK1-#     12/7 T
```

Mit diesem Beispiel lassen sich z.B. rund 341 ms erreichen; es werden 9 Byte Programmspeicher benötigt.

Eine elagante Lösung für eine Zeitschleife bietet die Verwendung von Blockverschiebebefehlen, wobei in der Variante

```
LD BC, Ø         10 T
PUSH HL          11 T
POP DE           10 T
LDIR             21/16 T   7 Byte
```

550,5116 ms erreicht werden können.

Im Registerpaar BC wird die Anzahl der Durchläufe angegeben. Da nach der PUSH/POP-Operation HL und DE den gleichen Inhalt haben, erfolgt die Blockverschiebung mit der Distanz Ø, d. h. die Speicherinhalte werden jeweils auf sich selbst geschrieben, was praktisch einem Lehrbefehl entspricht.

Bei all diesen Schleifen ist zu beachten, daß mindestens ein Registerpaar im Wert geändert und deshalb ggf. im Stack gerettet werden muß. Beim Lernsystem ist zu beachten, daß durch den Anzeige-NMI zusätzliche Zeit benötigt wird.

Zeitschleifen haben generell den Nachteil, daß sie den Rechner voll zeitlich auslasten und somit keine andere Operationen möglich sind. Bei einem Uhrenprogramm würde das bedeuten, daß der Rechner außer der Uhrenfunktion keine anderen Rechnungen ausführen kann. Deshalb wird bei der Realisierung von Uhren in der Regel auf externe Möglichkeiten zurückgegriffen, wofür sich beim System U 880 besonders der CTC U 857 anbietet. Dessen Programmierung und Verwendung für den Interruptbetrieb wie auch für den Pollingbetrieb wurde im Punkt 5. behandelt.

- Geben Sie eine Zeitschleife an, welche eine Verzögerung von 1 s bewirkt (Einfluß des NMI bleibt unberücksichtigt).
- Geben Sie eine Programmfolge an, welche unter Verwendung eines CTC (Adressierung entsprechend Lernsystem) eine Zeitbasis von 1 s bewirkt.

b) Zeiteinheitenzähler

Die einzelnen Zeiteinheiten (Zehntelsekunden, Sekunden, Minuten, Stunden usw.) müssen in geeigneter Weise gezählt werden. Wird dafür der für eine Zeiteinheit typische Grenzwert erreicht, so ist dieser rückzusetzen und die nächsthöhere Zeiteinheit muß erhöht werden. So ist der Sekundenzähler von Ø bis 59 zu erhöhen, wobei auf den Wert 59 wiederum die Ø folgen muß. In diesem Fall ist dann der Minutenzähler zu erhöhen.

<u>Beispiel:</u>

```
LD A, (MINUTE)
INC A
LD (MINUTE),
CPM 6Ø              60 DEZIMAL
RNZ                 Z-FLAG IST Ø
```

```
LD A, Ø
LD (MINUTE), A
RET                    Z-FLAG IST 1
                       16 Byte      48/72 Takte
```

Um Speicherplatz und Zeit zu sparen, bietet sich folgende Befehlsfolge an:

```
LD A, (MINUTE)
INC A
LD (MINUTE), A
SUB 6Ø
RNZ
LD (MINUTE), A
RET
                       14 Byte      48/65 Takte
```

Anstelle des SUB-Befehls könnte prinzipiell auch ein ADD- oder ADC-Befehl entsprechend verwendet werden. In beiden Fällen wird mit Dualzahlen gearbeitet, welche je nach Art der Anzeige noch entsprechend umkodiert werden müssen. Da der U 880 eine BCD-Arithmetik erlaubt, kann für entsprechende Anwendungen direkt mit BCD-Zahlen gerechnet werden.

- Wie ist obiges Beispiel zu ändern, um direkt mit BCD-Zahlen zu arbeiten?

- Darf dabei mit dem Befehl INC A gearbeitet werden; bzw. was ist dabei zu beachten?

```
                    ORG ØFØØØH        Uhrenprogramm für das Lern-
                                      system unter Berücksichti-
                                      gung der für den Anzeige-
                                      NMI benötigten Zeit
Ø1 ØØ 6Ø ANFANG     LD BC,6ØØØØH    ; Wert für Zeitschleife -
                                              1ØØØØ H
54                  LD D, H         ; D = H
5D                  LD E, L         ; E = L
ED BØ               LDIR            ; Zeitverzögerung
ED BØ               LDIR
21 ØØ F2            LD HL,ØF2ØØH    ; Adresse Sekundenspeicher
Ø6 6Ø               LD B,6ØH        ; Grenzwert Sekunden u. Minu-
                                                          ten
CD ØØ F1            CALL INCT       ; UP Erhöhen Zeiteinheit
CC ØØ F1            CAZ  INCT       ; UP Erhöhen, wenn Sekunden-
                                                     Überlauf
Ø6 24               LD B,24H        ; Grenzwert Stunden
CC ØØ F1            CAZ  INCT       ; UP Erhöhen Zeiteinheit,
                                    ; wenn Minutenüberlauf
21 ØØ F2            LD HL,ØF2ØØH    ; Adresse Sekundenspeicher
7E                  LD A,(HL)       ; Sekundenwert
CD DC Ø1            CALL DISDAT     ; Sekunden darstellen
                                    ; (LS-Programm)
23                  INC HL
5E                  LD E,(HL)       ; Minuten
23                  INC HL
56                  LD D,(HL)       ; Stunden
EB                  EX DE, HL       ; Nach HL Laden für Eintritt
                                    ; in DISADR
CD E8 Ø1            CALL DISADR     ; Stunden und Minuten dar-
                                    ;stellen (LS-Programm)
C3 ØØ FØ            JP ANFANG
                    ORG ØF1ØØH
7E      INCT:       LD A,(HL)       ; Zeiteinheit laden
C6 Ø1               ADD,1           ; Um 1 erhöhen
27                  DAA             ; BCD-KOR.
77                  LD (HL), A      ; Neuen Wert speichern
9Ø                  SUB B           ; Test, ob Überlauf
2Ø Ø1               JRNZ NEXT-#     ; Sprung, wenn nicht
```

```
77              LD (HL), A      ; Einheit = Ø
                                ; Z-Flag ist gesetzt
23  NEXT:       INC HL          ; Zellenzeiger auf nächste
                                ; Zelle
C9              RET
```

- Ändern Sie das Programm so ab, daß es in einer Interrupt-Routine verwendet werden kann!

7. Das Mikrorechnersystem MRS K 1520

Das Mikrorechnersystem MRS K 1520 besteht aus einer Reihe modularer Komponenten der Gerätetechnik und den dazugehörigen Systemunterlagen (SUL).

Die Mikrorechner des MRS K 1520 sind auf folgenden Gebieten einsetzbar:

- automatisierte Produktionssteuerung
- Labor- und Prüffeldautomatisierung
- Informationsverarbeitungssystem
- wissenschaftlich-technische und ökonomische Aufgaben

Durch entsprechende Auswahl einzelner Komponenten der Gerätetechnik und Systemunterlagen sowie unter Hinzunahme handelsüblicher peripherer Geräte oder spezieller kundeneigener Einrichtungen sind Anwendungssysteme realisierbar, wie sie für die genannten Einsatzgebiete benötigt werden.

7.1. Gerätetechnik des MRS K 1520

Zur Gerätetechnik gehört ein Sortiment konstruktiv und funktionell abgestimmter Bau- und Funktionsgruppen, zu denen bestückte Steckeinheiteneinsätze usw. zählen.

Gerätekomponenten K 1520:

Bezeichnung	Kurzbez.	Chiffre	Technische Hauptparameter
Zentrale Recheneinheit	ZRE	K 2521	mit - Zähler/Zeitgeber - Taktgenerator - Anfangslöschung - parallel - E/A-Interface - Zentralprozessor U 880 - 4 k Byte Speicher (1 k stat. RAM 3 k EPROM)
		K 2511	wie K 2521, ohne Taktgenerator/ Anfangslöschung
		K 2523	wie K 2521, ohne Zähler/Zeitgeber

	K 2524	wie K 2522, ohne Zähler/Zeitgeber
	K 2525	mit - Zähler/Zeitgeber - Taktgenerator - Anfangslöschung - Zentralprozessor U 880 - 8 K Byte EPROM
Operationsverstärker OPS		
	K 3520	n-MOS 4 k Byte statischer RAM
	K 3521	C-MOS 4 k Byte statischer RAM
	K 3525	n-MOS 16 k Byte dynamischer RAM
Operativ-/Festwertspeicher OFS		
	K 3620	n-MOS 2 k Byte stat. RAM/ bis 6 k Byte EPROM
	K 3621	C-MOS 2 k Byte stat. RAM/ bis 6 k Byte EPROM
Festwertspeicher PSF		
	K 3820	bis 16 k Byte EPROM
Busverstärker BVE	K 4120	zur Systembusverlängerung und als Adapter für Prüfrechner
Anschlußsteuerung für Folienspeicher AFS	K 5121	Anschluß von max. 4 Floppy-Disk-Laufwerken des Typs MF 3200
Anschlußsteuerung für robotron-Geräte ADA	K 6022	1 Ausgabekanal) für Anschluß von 1 Eingabekanal) Geräten mit SIF 1000
Anschlußsteuerung für Bedieneinheit ABD	K 7022	zum Anschluß der BDE K 76 22

Bezeichnung	Kurz-bez.	Chiffre	Technische Hauptparameter
Bedieneinheit	BDE	K 7622	für Inbetriebnahme und Service
Anschlußsteuereinheit V.24	ASV	K 8021	2 duplexfähige Kanäle für Schnittstelle V.24 nach CCITT
Steckeinheiteinsatz	STS	K 0120	120 mm breit für 5 STE mit Systembus in Form gedruckter Rückverdrahtung
		K 0121	240 mm breit für 11 STE --"--
PROM-Programmiereinrichtung	PPE	K 0420	Programmierung und Prüfung von PROM's des Typs U 555
EPROM-löscher	PLG	K 0421	für gleichzeitiges Löschen von 8 EPROM's
PROM-Aufnahmeeinheit	PAE	K 0422	zum griffseitigen Anschluß an PPE K 0420
Verbindungsleitung	LTG	K 0521	für griffseitigen Anschluß des BVE K 4120
Adapter für Verbindungsleitung	VLA	K 0522	zur Kopplung des BVE K 4120 mit dem Sekundärbus
Anschlußsteuerung Bildschirm	ABS	K 7023	zum Anschluß der Monitore K 7221.10, K 7221.20 - mit der ABS K 7023.10 die Monitore K 7221.11, K 7221.21

7.2. K 1520 Systemunterlagen

7.2.1. Betriebssystem

Für die Geräte auf Basis K 1520 gibt es je nach Anwendungszweck verschiedene Betriebssysteme (BS).

7.2.1.1. Betriebssystem für Mikrorechnerentwicklungssysteme

Für das Mikrorechnerentwicklungssystem MRES vom VEB Robotron-Elektronik Zella-Mehlis existiert ein Betriebssystem MEOS (mit dem Steuerkern MEEX).
Dieses BS ist für die Entwicklung von Systemunterlagen für Geräte auf Basis K 1520 gedacht. Das BS MEOS umfaßt folgende Komponenten:

Steuerkern (MEEX) - Ansteuerung und Datenaustausch mit den angeschlossenen peripheren Geräten

Mikroprogrammaufbereitungssystem - bestehend aus Editor, Assembler, Linker für Mikroprogramme der SYPS K 1520 Assemblersprache. Dazu gehört noch ein Absolutassembler.

Editor für freies Format-Programmerstellung in einer Mikrosprache (BASIC) oder Texterfassung

BASIC-Interpreter

Testmonitor (TEMO) - zur komfortablen Testung von Mikroprogrammen im MRES oder in einem gekoppelten Anwendersystem

Service-Programme - z. B. zur Verwaltung von externen Datenträgern

7.2.1.2. BS für Datenverarbeitung

Für die Datenverarbeitungsgeräte (Bürocomputer, Datensammelsystem, Buchungs- und Fakturierautomaten) auf K 1520-Basis von den Robotronbetrieben BWS und BWK wurde ein BS entwickelt, das auf dieses Anwendungsgebiet zugeschnitten ist.
Für dieses BS SIOS mit dem Steuerkern SIEX existiert auch eine Makrobefehlssprache MABS 1520, die zur anwenderfreundlichen Programmierung der Bürocomputer (z. B. A 5120, A 5130) gedacht ist. Zur Programmaufbereitung von Programmen in MABS 1520 bzw. deren Teilsprache MABS 1520 SUBSET (für BC A 5110) gibt es entsprechende Editor-Linker- und Assemblerprogramme. Ebenso Mischassembler zwischen den Sprachen SYPS K 1520 und MABS 1520.

7.2.1.3. BS zur Steuerung von Echtzeitprogrammen

Für den Anwender, der mit Hilfe von K 1520-Modulen Echtzeitprozesse steuern will, gibt es die Echtzeit-BS EIEX.

Grundvariante des EIEX ist sein Steuerkern mit den Komponenten:
- Interruptbehandlung
- Vorrangsteuerung
- Prüf- und Kommandoorganisation
- Echtzeituhr

Desweiteren gibt es drei gleichrangige Erweiterungsstufen:
1. Komplex:
   - Bedienerkommunikation (Tastatur, Display, Fehlerorganisation)
   - UP-Technik (Steuerung/Verwaltung von Anwender-UP)
2. Komplex:
   - DV-Peripherie (Drucker, LBL, LBS, FD, KMB, Display usw., dabei sind je 8 Geräte zulässig)

3. Komplex:
   - Prozeßperipherie (URSADAT 5000)

Das Echtzeit-BS EIEX ist so angelegt, daß sich der Anwender für sein spezielles Problem ein optimales BS zusammenstellen kann. Dieses wird mit dem speziellen Programmaufbereitungssystem EMOS, bestehend aus
   - Editor EDITE
   - Assembler ASSE
   - Generierungsprogramm GENE
   - und Binder LINKE,

generiert. EMOS läuft nur auf dem MRES. Anwender-EIEX-BS werden im allgemeinen vom RVB generiert.

7.2.2. Basis-MOS (MOS = Maschinenorientierte Systemunterlagen)

Im Rahmen der Basis-MOS wurden Geräteroutinen als systemunabhängige Programmbausteine für den Anschluß einiger peripherer Geräte an den Mikrorechner K 1520 sowie Standardmodulen, d. h. Programme zur rechentechnischen Anwendung der arithmetischen Grundfunktionen und mathematischen Standardfunktionen, erarbeitet.

Die Nutzung dieser Gerätroutinen und Standardmodulen erleichtert den Einsatz von Mikrorechnern, da der Anwender damit erheblichen Programmieraufwand einsparen kann.

7.2.2.1. Geräteroutinen für:
   - Lochbandleser 1210
   - Lochbandstanzer 1215
   - Seriendrucker 1156
   - Kassettenmagnetband K 5200
   - Folienspeicher MF 3200
   - PROM Programmiereinrichtung K 0420
     (nur für U 555 und Äquivalenztypen)

- DEKK-Tastatur K 7602
  (Typ AN 2)
- V.24-Übertragungsprozedur AP 62/64

Dieses Programm dient dem Betrieb des K 1520 als abhängige Datenendeinrichtung (DEE) an steuernden DEE (Master-Slave-Prinzip) über die bitseriellen Schnittstellen I 2 (CCITT V.24) oder IFFS im asynchronen Halbduplexbetrieb.

7.2.2.2. Standardmoduln

Die Standardmoduln dienen der rechentechnischen Bearbeitung von wissenschaftlich-technischen Aufgaben. Sie umfassen die arithmetischen Grundoperationen, einige mathematische Standardfunktionen sowie die erforderlichen Konvertierungsprogramme und werden für drei unterschiedliche Datenformate
- Gleitkommaformat
- Festkommaformat
- BCD-Format

angeboten.
Außerdem existieren Programme zur wechselseitigen Konvertierung von Gleitkomma- und Festkommazahlen in interner Darstellung.

Die Standardmoduln für das Gleitkommaformat beinhalten die mathematischen Funktionen. Für geringere Ansprüche und für einen beschränkten Zahlenbereich ist das Festkommaformat geeignet. Vorteil: kürzere Rechenzeiten.

Die BCD-Moduln können hauptsächlich bei der Lösung ökonomischer Aufgaben benutzt werden.

7.2.3. Cross-Systemunterlagen

Für die Entwicklung und Testung von K 1520-Programmen auf Großrechnern (R 4000, R 4200) gibt es umfangreiche Systemunterlagen:
- Cross-Assembler
- Cross-Simulations- u. Testsystem
- Cross-Binder
- Cross-Korrekturprogramm

Weiterhin gibt es noch diverse Cross-MOS für ESER-Rechner.

8. Literaturverzeichnis

(1) H. Kieser, M. Meder: Mikroprozessortechnik - Aufbau und Anwendung des Mikroprozessorsystems U 880 D; Verlag Technik Berlin 1982, 352 Seiten, 36.-
Neben einer ausführlichen Beschreibung des Systems U 880 D wird insbesondere auf das U 880-Lernsystem und die FPS2 eingegangen.

(2) W. Schwarz, G. Meyer, D. Eckhardt: Mikrorechner-Wirkungsweise, Programmierung, Applikation; Verlag Technik Berlin 1980, 360 Seiten, 32.-
Nach einer kurzen Darstellung der Grundlagen werden verschiedene Mikroprozessoren vorgestellt. Für diese werden eine Vielzahl von Programmbeispielen aufgeführt und in verschiedenen Varianten diskutiert.

(3) A. Jugel: Mikroprozessorsysteme; Verlag Technik Berlin 1978, 204 Seiten, 20.-
Dieses Buch dient hauptsächlich der Beschreibung der Grundlagen und der Hardware von Mikrorechnern.

(4) M. Roth: Mikroprozessoren, Wesen - Technologie - Weiterentwicklung, Aufbau - Programmierung - Anwendung; Wissenschaftliche Zeitschrift der Technischen Hochschule Ilmenau
4. Auflage 1979, 256 Seiten, 17.-
Nach einer Beschreibung der technologischen Grundlagen werden eine Vielzahl von Mikroprozessoren vorgestellt, ebenso weitere für den Aufbau von Mikrorechnern benötigte Schaltkreise. Weiterhin wird auf die Mikrorechnersysteme robotron K 1510 und K 1520 sowie auf verschiedene Entwicklungssysteme, wie z. B. das MRES und MICROCOMBI eingegangen.

(5) L. Claßen: Programmierung des Mikroprozessorsystems U 880 - K 1520 (Reihe AT, Band 192)
Verlag Technik Berlin, 2. Auflage 1982, 79 Seiten, 4.80
Das Buch enthält eine kompakte Beschreibung des Mikroprozessorsystems U 880 D, insbesondere der U 880-Assemblersprache und der Programmierung der peripheren Schaltkreise

(6) Oetker/Claßen: Mikroprozessor-Betriebssysteme (Reihe AT, Band 201);
Verlag Technik Berlin

(7) Autorenkollektiv: Softwaretechnologie für Mikrorechner, Verlag Die Wirtschaft, etwa 12.-

(8) H. Barthold, H. Bäurich: Mikroprozessoren - Mikroelektronische Schaltkreise und ihre Anwendung, 3 Teile (electronica 186 - 188);
Militärverlag der DDR 1980, je 1.90
Während sich Teil 1 mit den Grundlagen der Mikrorechentechnik beschäftigt, geht Teil 2 auf die Mikroprozessoren U 808, U 880 sowie den Intel 8080 ein. Teil 3 enthält die Beschreibungn der peripheren Schaltkreise sowie Beispiele für die Programmierung. Eine neue Auflage erschien mit den Heftnummern 202 bis 204.

(9) Technik der Mikrorechner, Reihe, in: radio fernsehen elektronik 26 (1977), H. 17 bis 28 (1979), H. 12
Die Reihe geht ausführlich auf Grundlagen, Hardware, Programmierungstechnik undMikrorechneranwendung ein.

(10) Technische Beschreibungen

- Zentrale Verarbeitungseinheit CPU U 880 D
- Schaltkreis für parallele Ein- und Ausgabe PIO U 855 D
- Schaltkreis für serielle Ein- und Ausgabe SIO U 856 D
- Schaltkreis für Zähler- und Zeitgeberfunktion CTC U 857 D

VEB Funkwerk Erfurt

(11) Befehlsbeschreibung U 880 D
VEB Mikroelektronik Karl Marx Erfurt

V 4 56 R 122 84